Introduction to IBM Data Processing Systems

Student Text

IBM

Student Text

Introduction to

IBM Data Processing Systems

Contents

Preface

All IBM Data Processing Systems, regardless of size, type, or basic use, have certain common fundamental concepts and operational principles. This manual presents these concepts and principles as an aid in developing a basic knowledge of computers. The manual is designed for use in training programs where a basic knowledge of computers is the end objective or is a prerequisite to the detailed study of a particular IBM system.

Each section is organized to present a logical association of related concepts and operational principles. The sections may be used in a progressive sequence to develop a concept of the computer system, or they may be used independently as reference material. The subject matter has been generalized and refers to actual machines and systems as little as possible. Specific systems are mentioned only to illustrate a general principle, not to compare one system with another.

Throughout this manual you will be reading about data processing concepts and devices supported by IBM. Such specifics as core requirements, device capacities and speeds, and special features will be discussed. However, because of the dynamic nature of data processing, where changes and improvements are being made at a very rapid pace, the reader is advised to refer to the IBM Systems Reference Library and other IBM publications for the most current information.

SECOND EDITION — (May 1968)
This edition is a minor revision of C20-1684-0 and does not obsolete it. Continued changes in the IBM product line are reflected in this edition.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601.

Technological advance in data processing is both dynamic and extensive. The ways in which data processing systems can be used seem almost boundless. Each new application demonstrates how such systems can be used to help man enlarge his capabilities.

Data processing systems ordinarily consist of a combination of programs and physical equipment designed to handle business or scientific data at electronic speeds with self-checking accuracy. The physical equipment (Figure 1) consists of various units, including input, storage, processing, and output devices. Figure 2 pictures a teleprocessing (telecommunications plus data processing) system applied to airline reservation activities.

Machines are devised by men for a purpose. In the case of data processing machines, the purpose can be expressed simply: they offer man a means to increase his productivity.

They do this in two ways. First, they enable man to increase his output per hour and the quality of his output (this is true whether it be in research, production, problem solving, or the distribution of goods and services). Second, they increase productivity by encouraging careful and intelligent planning.

Data processing machines came into being primarily to meet the increased need for information under increasingly complex conditions.

As a manufacturing economy developed during the

Figure 1. IBM System/360 Model 40 Data Processing System

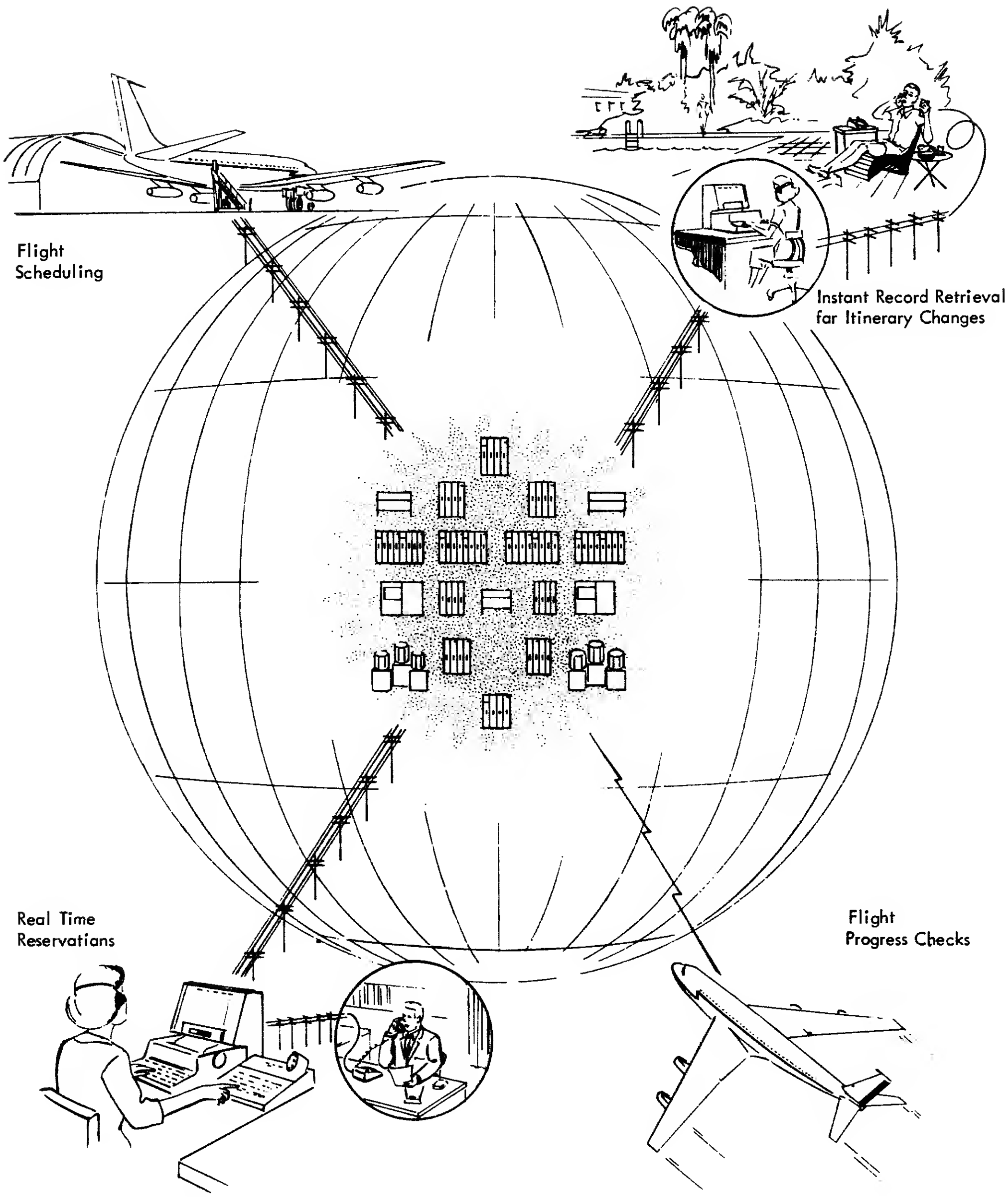


Figure 2. Data processing system application

19th century, it became clear that expanded markets would require mass production techniques. Machinery was introduced to increase productivity. It became possible to turn out more and more goods with less human effort.

During the last quarter century, further changes have taken place. Science has moved into the forefront of human activity. Research has grown to a multibillion-dollar-a-year undertaking. New technology has provided a new impetus for corporate growth. Service industries have multiplied. Patterns of consumer spending have changed.

As these changes gained force, they manifested themselves in many ways. Informational needs greatly increased. Data assumed new importance. Clerical tasks multiplied. It seemed that paper handling alone would overwhelm all productive activities, for clerical mechanization had not kept pace with production line developments in the factory.

Great opportunities and challenges lie ahead. An example of what can be done is the development a few years ago of magnetic character sensing for the banking industry. The estimated 10 billion checks that circulate annually in the United States present a staggering task in data handling for banks. Each check drawn on a bank must be handled at least six times before it

is canceled and returned. Even when business machines were introduced to handle part of this chore, operators were needed to transfer data from the checks to a form in which the data could be used by the machines.

Magnetic character sensing, developed by computer manufacturers in cooperation with the American Bankers Association (ABA), permits data to be read directly by both man and machine (Figure 3). By agreement among computer manufacturers, check printers, and the ABA, such banking documents as checks, deposit slips, and debit and credit memos can be printed in magnetic ink. Printed information about the bank of origin, depositor's account number, and other essential data can be read directly by the machine. Only the specific amount of each check or deposit slip need be recorded on the document in magnetic print, and this need be done only once by an operator to process the document through its entire routine.

In addition to the growing need for mechanization of clerical routines and management procedures, there is the tremendously expanded need for data processing to match the new rate of technological growth and scientific research. The demands for information are enormous. Data processing systems are increasingly relied upon for information to assist in running enterprises, administering institutions, directing research, and planning future activities. To this end, data processing centers are, increasingly, offering time-sharing services to their users. The users can enter problems to be solved, requests for information, and data to be processed — all from remote terminals located either on site or possibly thousands of miles away. The automatic reservation systems for airlines and motels are examples of the long-distance entries in intracompany time sharing.

Two other areas of remarkable advances are image processing and audio response. The processing of the Mariner IV pictures transmitted from the planet Mars to Earth was an example of image processing.

Rapid microfilm scanning was combined with automatic interpretation of dark and light spots into 1's and 0's for computer storage. These pictures were then displayed on viewers' screens.

Although it is possible, experimentally, to dictate or speak directly to a binary recording device that will compute from the dictation, the converse is being done daily at the New York Stock Exchange. There, the latest stock prices are quoted on request, by a recorded voice. The message is selected and assembled from spoken words previously stored in the computer.

Regardless of the product or problem, the nature of the enterprise or institution, wherever there is need for information upon which human judgments can be based, there may also exist a need for a data processing system.

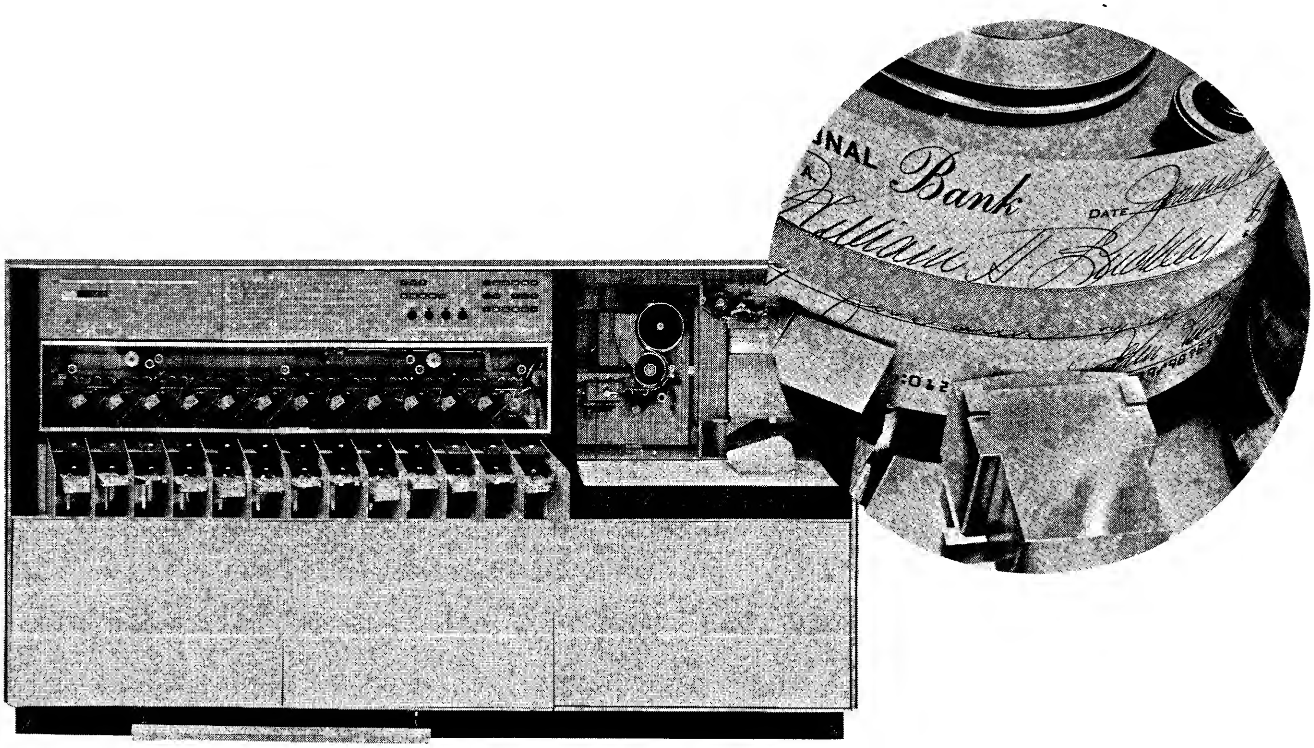


Figure 3. Magnetic character sensing — IBM 1419 Magnetic Character Reader

Data Processing in the Past

Although data processing systems are tools of astonishing versatility, the automatic processing of data is so recent that its biggest period of growth can be traced within the last 40 years.

Punched cards were introduced during the census of 1890, but the data processing industry, as recently as 1930, amounted to little more than a fledgling.

World War II caused a swift change of pace in data processing developments. Much of the momentum came from the urgent demands of science. In aircraft design and ordnance development, new and prodigious requirements for data were encountered. As work got under way on the atomic bomb, scientists found themselves suddenly faced with new dimensions in calculation.

Both here and abroad, the first two large-scale computers were developed in university laboratories. The earliest, the ENIAC, came from the University of Pennsylvania; Europe's first, the EDSAC, came from the laboratories of Cambridge University, in England.

In these machines, the switching and control functions, once entrusted to relays, were handled by vacuum tubes. Thus, the relatively slow movements of switches in electromechanical computers were replaced by the swift motion of electrons. By this changeover, it became possible to increase the speed of calculation and perform computations 1000 times as fast as before.

Almost concurrently with the use of electronics came another major development that was to widen the capabilities of data processing systems and expand their opportunities for application. This advance is embodied in what is called a stored program computer. At the start, machine instructions were programmed on interchangeable control panels, cards, or paper tapes. Detailed instructions had to be wired in or read into the machine as the work progressed. Data put into the computer was processed according to the instructions contained in these preset devices. Only in a limited way could the computer depart from the fixed sequence of its program.

It soon became apparent that these programming techniques inhibited the performance of the computer. To give the computer greater latitude in working problems without operator assistance, scientists proposed that the computer store its program in a high-speed internal memory or storage unit. Thus, the computer would have immediate access to instructions as rapidly as it called for them. With an internal storage system, the computer could process a program in much the same way that it processed data. It could even be made to modify its own instructions as dictated by developing stages of work.

The earliest computer to incorporate this feature was completed in 1948. Later computers extended the principle until it became possible for a computer to generate a considerable part of its own instructions.

Because the computer is capable of making simple decisions, and because it is capable of modifying instructions, the user is relieved of a vast amount of costly and repetitive programming.

Concurrently with the development of stored programs for computers, teleprocessing was being born, although it was not known by that name for more than a decade. In 1940, the U.S. Air Corps voiced a need for a machine to automatically punch IBM cards with the data received over telegraph lines in the form of punched paper tape. To answer this need, IBM produced a tape-controlled card punch and a card-controlled tape punch. During the last two years of World War II, 4 to 5 million cards per month were transmitted from point to point by telegraph.

The next major advance in teleprocessing was the introduction of the IBM Data Transceiver (1954) to provide direct card-to-card transmission over voice-grade (telephone) channels, as well as microwave, short-wave radio, and telegraph channels.

The early 1950s saw the introduction of medium- and large-scale data processing systems, specifically designed to take over the burdensome clerical chores that beset so many growing companies.

Though essentially similar to previous computers in the way they processed data, these new business systems differed substantially in various parts of their makeup. In scientific research, most problems call for relatively few items to be subjected to intensive machine processing. In business operations, the reverse is more often true. Here the need is for machines that accommodate vast numbers of items, while the processing, by comparison, is ordinarily quite simple.

Modifications in these new business systems were addressed to the twin problems of input and output.

Early computers had used punched cards and paper tapes for the input of information. Then a method was developed for storing information as magnetized spots on magnetic tape. This new technique provided input speed 50 to 75 times that of cards and brought improvement in input, output, and storage. More recent advances in magnetic tape technology have greatly increased the original input/output rate.

After the Korean War, man's need seemed to be constantly one jump ahead of the computer's ability to handle the logical and arithmetic labors of his reasoning. The demand quickened especially in such fields as nuclear physics and space technology, where work on the H-bomb and ballistic missiles presented problems that put a severe strain on the capacities of

existing machines. Still more speed was needed.

A substitute for earlier storage devices appeared in the early 1950s – the magnetic core, which is a small ring of ferromagnetic material. When strung on a complex of fine wires (Figure 4), magnetic cores can be made up into a high-speed internal storage system. An array of cores – some magnetized in one direction, some in the other – represents items of information. Items in a core storage can be located and made ready for processing in a few millionths of a second.

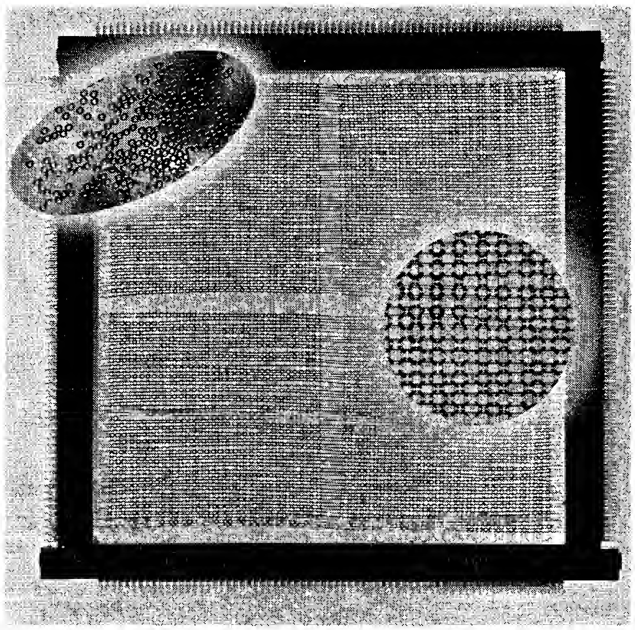


Figure 4. Magnetic core plane

Almost at the same time, other engineers developed magnetic drum storage. Access to information stored on the drum was substantially slower than with the core system, but storage capacity was substantially increased, and access was still faster than with magnetic tape.

Other conditions peculiar to business led to still more developments. A major one is a system that overcomes a problem – batching – often encountered in data processing. For example, magnetic tapes store information sequentially, and the user must accumulate information in batches before putting it on tape. Otherwise, the computer would be prohibitively costly and time consuming. But when this limitation is applied to business practice, it means that each item of information can be only as current as the batch in which it is bundled for delivery to the computer. In ordinary operations, hours and sometimes days may elapse between batches.

Because of the sequential nature of tapes, the limitation is compounded when the user calls for the retrieval of a piece of information. The computer is forced to search through a long reel of information for the piece. Access is slow; time may be lost.

Batching and searching requirements frequently present serious drawbacks, even in scientific work. In business, the difficulty becomes much more acute, especially in accounting procedures.

Inline data processing was provided in the mid-50s with the introduction of random (direct) access storage units. These allow direct access to the desired data record addressed, thus reducing the processing time required in sequential processing. The first direct access storage units consisted essentially of a stack of magnetically coated, rotating disks, each disk containing data tracks. Information can be recorded on, or retrieved from, the data tracks without regard to the sequential order of recorded data.

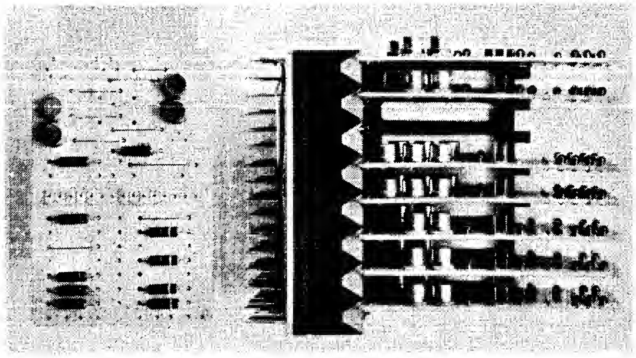
Meanwhile, continuing developments in electronics and solid state physics led to still newer and better components.

In some switching functions, the vacuum tube was replaced by a smaller semiconductor diode that has the advantage of demanding less power. A further advance came when tiny transistors were introduced in place of vacuum tubes in the computer. Not only can these transistors be packed into smaller units (Figure 5), but they have greater reliability. The changeover to transistors was accomplished, creating what has frequently been referred to as the "Second Generation" of computer.

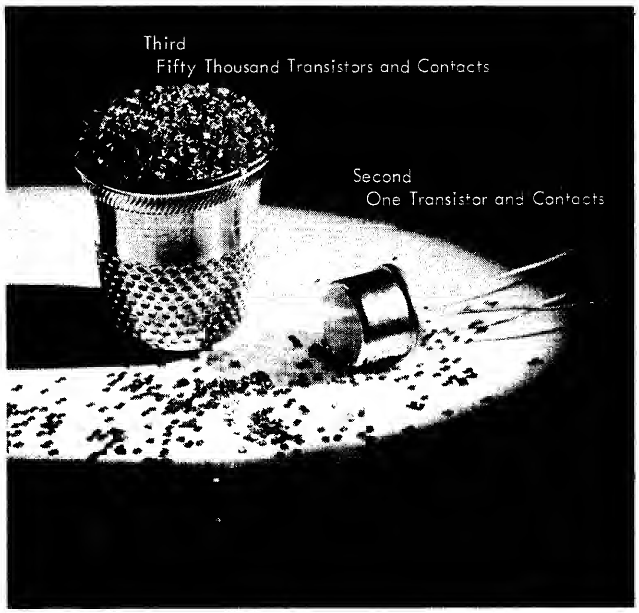
The next technological advance miniaturized and refined components of the Second Generation. This, when done, led to a concept known as Solid Logic Technology. The use of these components (Figure 5) ushered in the "Third Generation" of computer.

Research scientists have already advanced to still further stages in design. Some are studying the use of microwave phenomena as a medium for performing computer logic. Others are studying the behavior of materials and electrons at extremely low temperatures (cryogenics).

As always, the objective is to develop a better, more versatile, more useful computer – one that will work faster, store more information, require less power, occupy less space, and cost less.



Second Generation Components



Second and Third Generation Components

Figure 5. Computer components

Data Processing in the Future

Computers of the future will inevitably introduce changes in the way we work, in the way we learn, and even in the way we provide for our armed defense.

The new science of automatic programming seeks to make programming easier and more manageable. The goal is to build and program computers so that they accept instructions in a language close to English. Ultimately, computer scientists hope to develop machines that can read ordinary printed matter and that can respond to spoken words.

A significant advance in input/output technique is the development of the various types of graphic displays. These are similar in appearance to television sets, but the "picture" is computer output in the form of printed characters or graphic designs specified by the program and data. In some cases, the user can change the output display by using a "light pen" to "erase" a character (or part of a line) from the display screen. Then, by using the light pen or the associated keyboards, or both, he can alter the displayed information.

Computers of the future, as well as programs, will probably be quite different from those of today. Storage and processing units will be drastically reduced in physical size, yet speed and capacity will be greatly increased. Already many systems are serving a number of widely separated inquiry stations and remote terminals. Such systems of large networks have integrated widely scattered business operations. These physical changes have brought a need for control programming – to monitor the whole system – and a need for shorter operating programs and priority ratings for programs.

Information can be communicated and processed more accurately and with less cost by network-integrated data processing systems. Some even contemplate a time when paper bank checks will disappear in many transactions. Instead of handing an employee his paycheck, a paymaster some day may simply instruct a computer, serving a bank, to credit the employee's earnings to his bank account.

The Data Processing System

Data processing is a series of planned actions and operations upon information to achieve a desired result. The procedures and devices used constitute a data processing system (Figure 6). The devices may vary: all operations may be done by machine, or the devices may be only pencil and paper. The procedures, however, remain basically the same.

There are many types of IBM data processing systems. These vary in size, complexity, speed, cost, levels of programming systems, and application. But, regardless of the information to be processed or the equipment used, all data processing involves at least three basic considerations:

1. The source data or input entering the system
2. The orderly, planned processing within the system
3. The end result or output from the system

Input may consist of any type of data: commercial, scientific, statistical, engineering, and so on (Figure 7).

Processing is carried out in a preestablished sequence of instructions that are followed automatically by the computer (Figure 8). The plan of processing is always of human origin. By calculation, sorting, analysis, or other operations, the computer arrives at results that may be used for further processing or recorded as reports or sets of data.

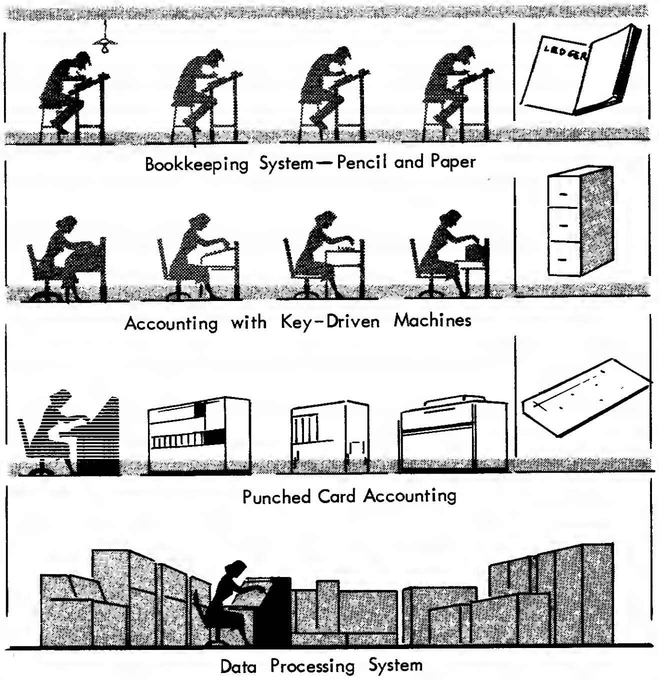


Figure 6. Data processing systems

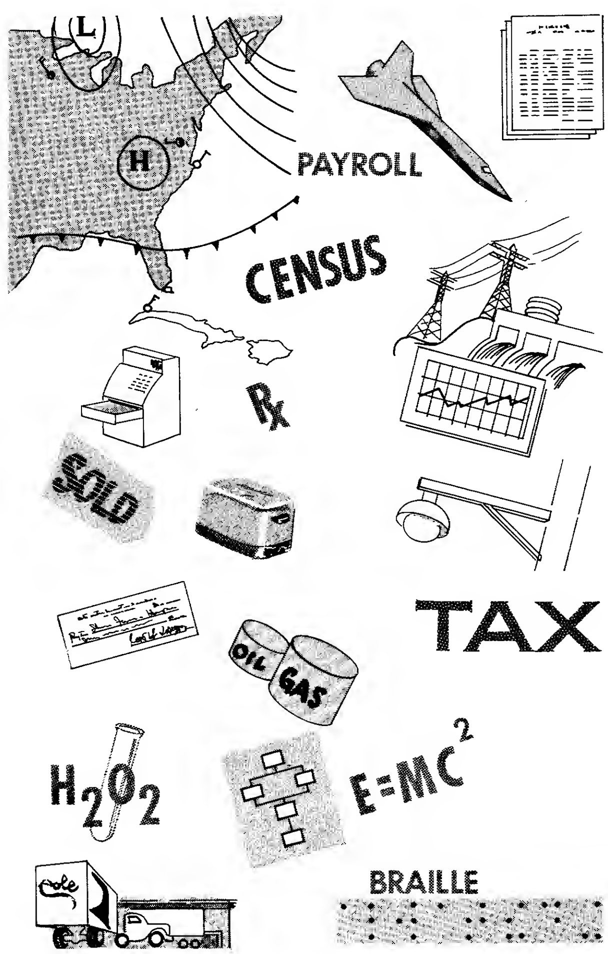


Figure 7. Sources of data

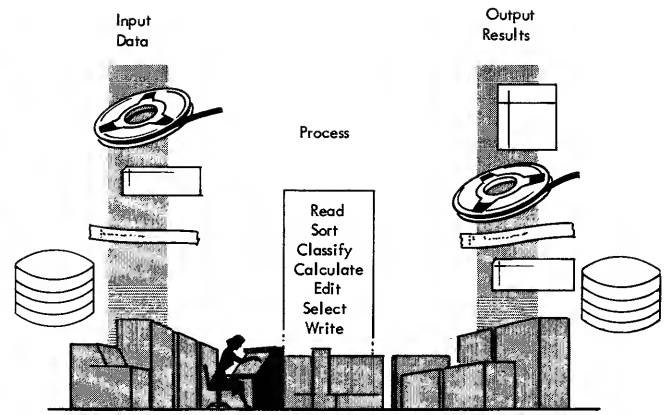


Figure 8. Data processing by computer

Stored Programs

Each data processing system is designed to perform a specific number and type of operations. It is directed to perform each operation by an instruction. The instruction defines a basic operation to be performed and identifies the data, device, or mechanism needed to carry out the operation. The entire series of instructions required to complete a given procedure is known as a program.

For example, the computer may have the operation of multiplication built into its circuits in much the same way that the ability to add is built into a simple desk adding machine. There must be some means of directing the computer to perform mutiplication, just as the adding machine is directed by depressing keys. There must also be a way to instruct the computer where in storage it can find the factors to multiply.

Further, the comparatively simple operation of multiplication implies other activity that must precede and follow the calculation. Assume that the multiplicand and the multiplier are read into storage by an input device. This device must previously have had access to the record or records from which these factors are to be supplied. Once the calculation is performed, the product must be returned to storage at a specified location, from which it may be written out by an output device.

Any calculation, therefore, involves reading, locating factors in storage, perhaps adjusting the result, returning the result to storage, and writing out the completed result. Even the simplest portion of a procedure involves a number of planned steps that must be spelled out to the computer if the procedure is to be accomplished.

An entire procedure is composed of these individual steps grouped in a sequence that directs the computer to produce a desired result. Thus, a complex problem must first be reduced to a series of basic

machine operations before it can be solved. Each of these operations is coded as one instruction or a series of instructions, in a form that can be interpreted by the computer, and is placed in the main storage unit as a portion of a stored program.

The possible variations of a stored program provide the data processing system with almost unlimited flexibility. A computer can be applied to a great number of different procedures simply by reading in or loading the proper program into storage. Any of the standard input devices can be used for this purpose, because instructions can be coded into machine language just as data can (see section titled "Machine Coding").

The stored program is accessible to the machine, providing the computer with the ability to alter the program in response to conditions encountered during an operation. Consequently, the program selects alternatives within the framework of the anticipated conditions.

A brief introduction to various types of programs and systems operations follows. All of the terms are discussed in greater detail later in the manual.

To make possible the teleprocessing networks and the orderly operation of many types of input/output devices that may be online with a computer, control programs have been developed by IBM and users of IBM computers. Control programs are also known as monitor programs or supervisory programs and they act as traffic directors for all the other programs. The others, called processing programs or problem programs, solve a problem or carry out a particular operation or process on a set of data and later relinquish control of the computer to the control program, which may be constructed to allow the computer to handle random inquiries from remote terminals, switch from one problem program within the computer to another, control external equipment or do whatever the application calls for.

The concept of maintaining optimum computer usage by interleaving and interspersing processing programs under the direction of control programs gives rise to the use of two terms – time sharing and multiprogramming.

Briefly, time sharing may be thought of as the cooperative use of a central computer by more than one user (company, division or branch of a company, institution, or government agency). Each user receives a share of the time available, with the result that many jobs are being performed within a congruent time (either simultaneously or seemingly simultaneously). This service may be achieved by interspersing programs rapidly on one computer system, by multiprogramming (described later), or by using two computers that are joined to permit the sharing of each other's facilities (multiprocessing).

Multiprogramming is usually thought of as a system of control programs and computer equipment that permits many processing or operating programs of one or more users to go on concurrently. This is accomplished by interleaving the programs with each other in their use of the central processing unit, storage, and input/output devices. To do this, the control programs and equipment must be able to identify the point at which a problem program that is being executed must "wait" for the completion of some event. At that point, the control program begins another processing task that is ready to be executed. When that is done, the control program must be able to go on to something else or go back to the former (unfinished) program, if it is ready to continue. Since many programs may be in stages of partial completion, successful multiprogramming usually requires scheduling levels of priority for the different tasks.

Time sharing, multiprogramming, and multiprocessing are closely linked, and may be combined in many ways. While one user has the computer on a time sharing basis, his problem may involve several different tasks that can be interleaved by a computer and programming system that provides for multiprogramming. It is also perfectly possible for teleprocessing messages to be coming in and going out of certain types of computers at the same time that process (problem) programs are being run. These are but two examples of possible combinations of time sharing and multiprogramming.

Functional Units

Data processing systems can be divided into four types of functional units: central processing unit, storage, input devices, and output devices.

Central Processing Unit

The central processing unit (Figure 9) is the controlling center of the entire data processing system. It can be divided into two parts:

1. The arithmetic/logical unit
2. The control section

The arithmetic/logical unit performs such operations as addition, subtraction, multiplication, division, shifting, moving, comparing, and storing. It also has a logical capability to test various conditions encountered during processing and to take action accordingly.

The control section directs and coordinates the entire computer system. Its functions involve controlling the input/output units and the arithmetic/logical operation of the central processing unit, and transferring data to and from storage, within given design limits. This section directs the system according to the procedure originated by its human operators and programmers.

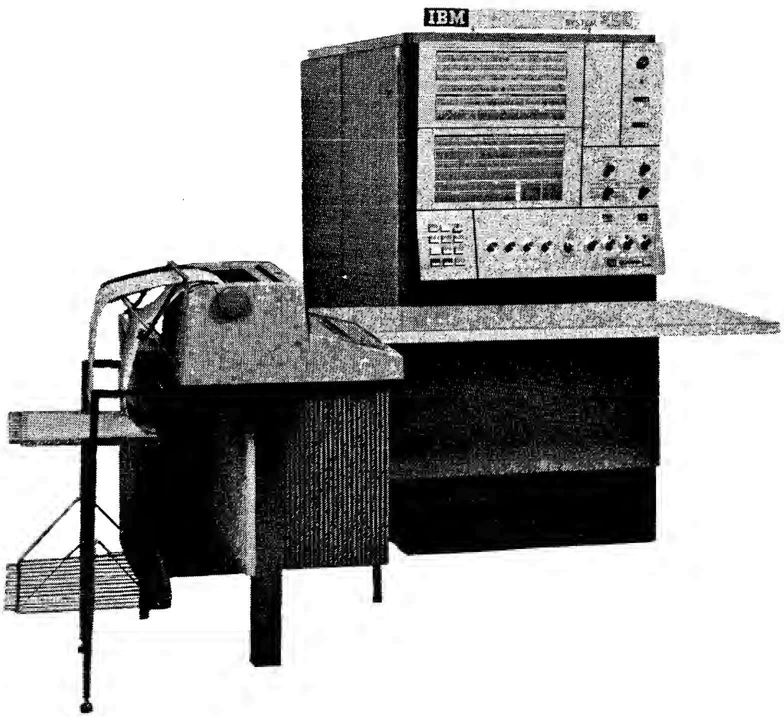


Figure 9. Central processing unit and console

Storage

Storage is somewhat like an electronic filing cabinet, completely indexed and instantaneously accessible to the computer.

All data must be placed in storage before it can be processed by the computer. Information is read into storage by an input unit and is then available for internal processing. Each position or section of storage has a specific location called an address, so that the stored data can be readily located by the computer as needed.

The computer may rearrange data in storage by sorting or combining different types of information received from a number of input units. The computer may also take the original data from storage, calculate new information, and place the result back in storage.

The size or capacity of storage determines the amount of information that can be held within the system at any one time. In some computers, storage capacity is measured in millions of digits or characters (bytes), providing space to retain entire files of information. In other systems, storage is smaller, and data is held only while being processed. Consequently, the capacity and design of storage affect the method in which data is handled by the system.

In System/360, main storage is thought of as consisting of the following: main data storage, which may vary in size from 4096 to over a million characters of programs and other data; control storage, which often contains special built-in "microprograms" to assist the computer in carrying out its own operations; local storage, consisting of high-speed working areas (registers) for floating-point arithmetic, fixed-point arithmetic, and other types of processing; and large-capacity storage (Figure 10), multiples of which can be added to the larger models, and which provides up to 8 million characters (bytes) of information.

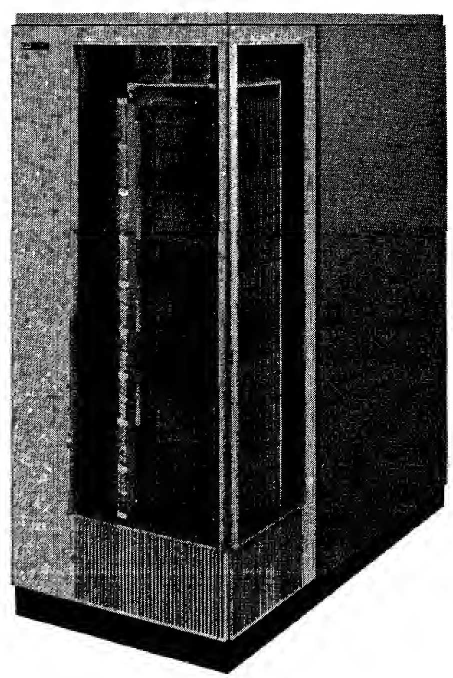


Figure 10. IBM 2361 Core Storage

In addition, much more storage is provided by the direct access storage devices described in a later section. These direct access devices and the tape units provide what is sometimes called secondary storage or auxiliary storage.

Storage is designed in such a way that information can be put there in many forms – as complete records, portions of records, digits, symbols, characters, code patterns, signals, and so on. However, capacity is usually stated in characters, meaning letters of the alphabet, digits, and special symbols of accounting, scientific notation, and report writing. In System/360, the word "byte" is used instead of "character". It is possible to "pack" two numeric digits into the same storage space that is required for letters of the alphabet, special characters, and the other symbols usually referred to as characters.

Input and Output Devices

The data processing system requires, as a necessary part of its information-handling ability, features that can enter data into the system and record data from the system. These functions are performed by input/output devices (Figure 11) linked directly to the system.

Input devices read or sense coded data that is recorded on a prescribed medium and make this information available to the computer. Data for input is recorded in cards and paper tape as punched holes, on magnetic tape as magnetized spots along the length of the tape, on paper documents as characters or line drawings created with the light pen and associated keyboards, etc.

The method of recording data for machine use and the characteristics of each medium are discussed in later sections.

Output devices record or write information from the computer on cards, paper tape, and magnetic tape; they print information on paper; generate signals for transmission over teleprocessing networks; produce graphic displays, microfilm images, and take other specialized forms. The number and type of input/output devices connected directly to the computer depend on the design of the system and its application.

Special data conversion operations are associated with all computer systems to transcribe information recorded on one medium to another. For example, information on punched cards can be transcribed automatically to magnetic tape. This operation may take place online, using the computer, or offline, using input/output devices independently.

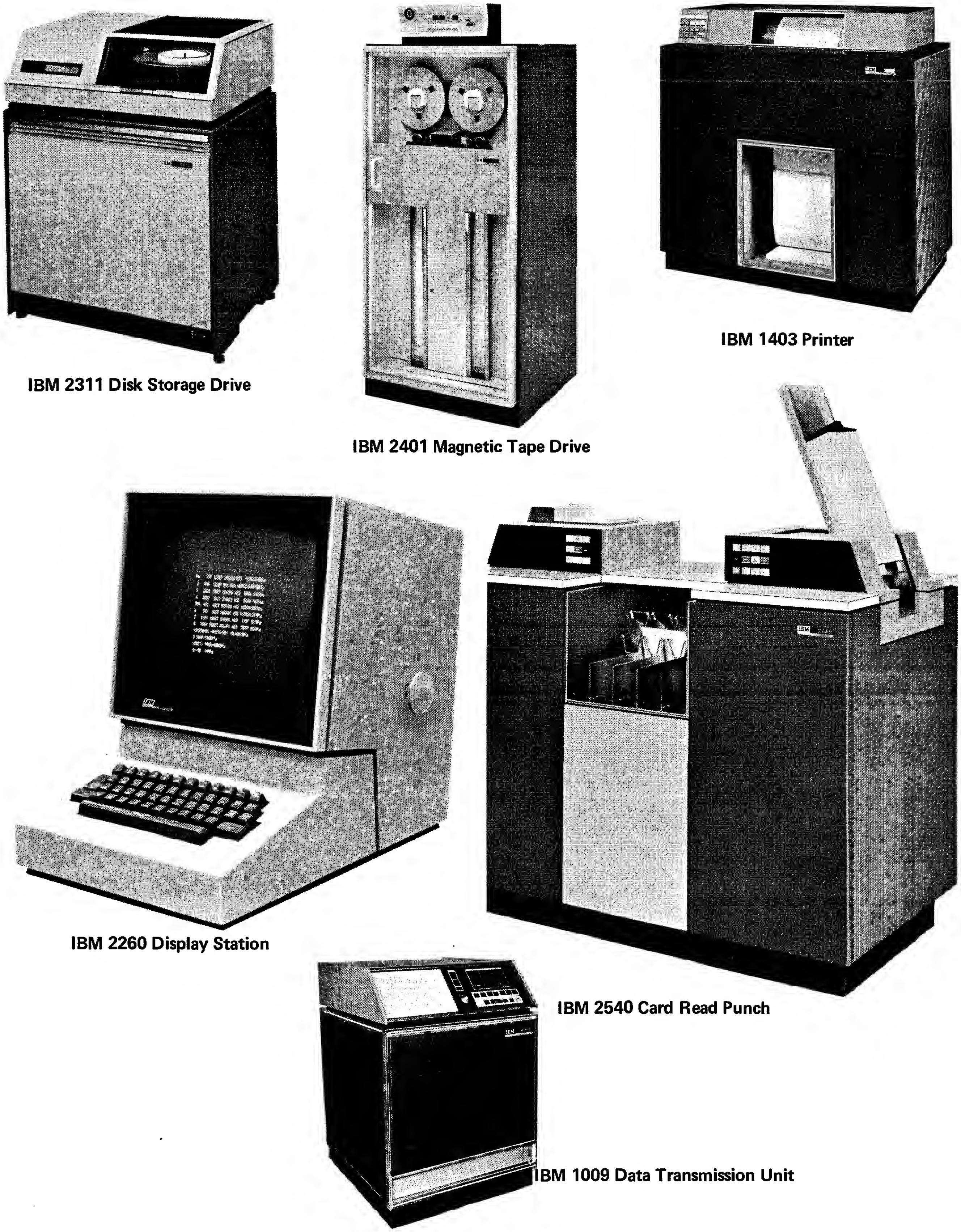


Figure 11. Input/output devices

Console

The console (Figure 12) is an input/output device that provides external control of the data processing system. Keys turn power on or off, start or stop operation, and control various devices in the system. Data may be entered directly by manually depressing keys. Lights are provided so that data in the system may be displayed visually.

On some systems, a console printer and keyboard provide limited output or input. The input/output device may print messages, signaling the end of processing or an error condition. It may also print totals or other information that enables the operator to monitor and supervise operation, or it may give instructions to the operator. On the other hand, it may be used to key in meaningful information (such as altering instructions) to a data processing system that is programmed to respond to such messages.

A remote console (Figure 13) may offer increased efficiency and flexibility by providing duplicate operator controls at a station removed from the processing unit.

Figure 13. IBM 2150 Console for remote operation

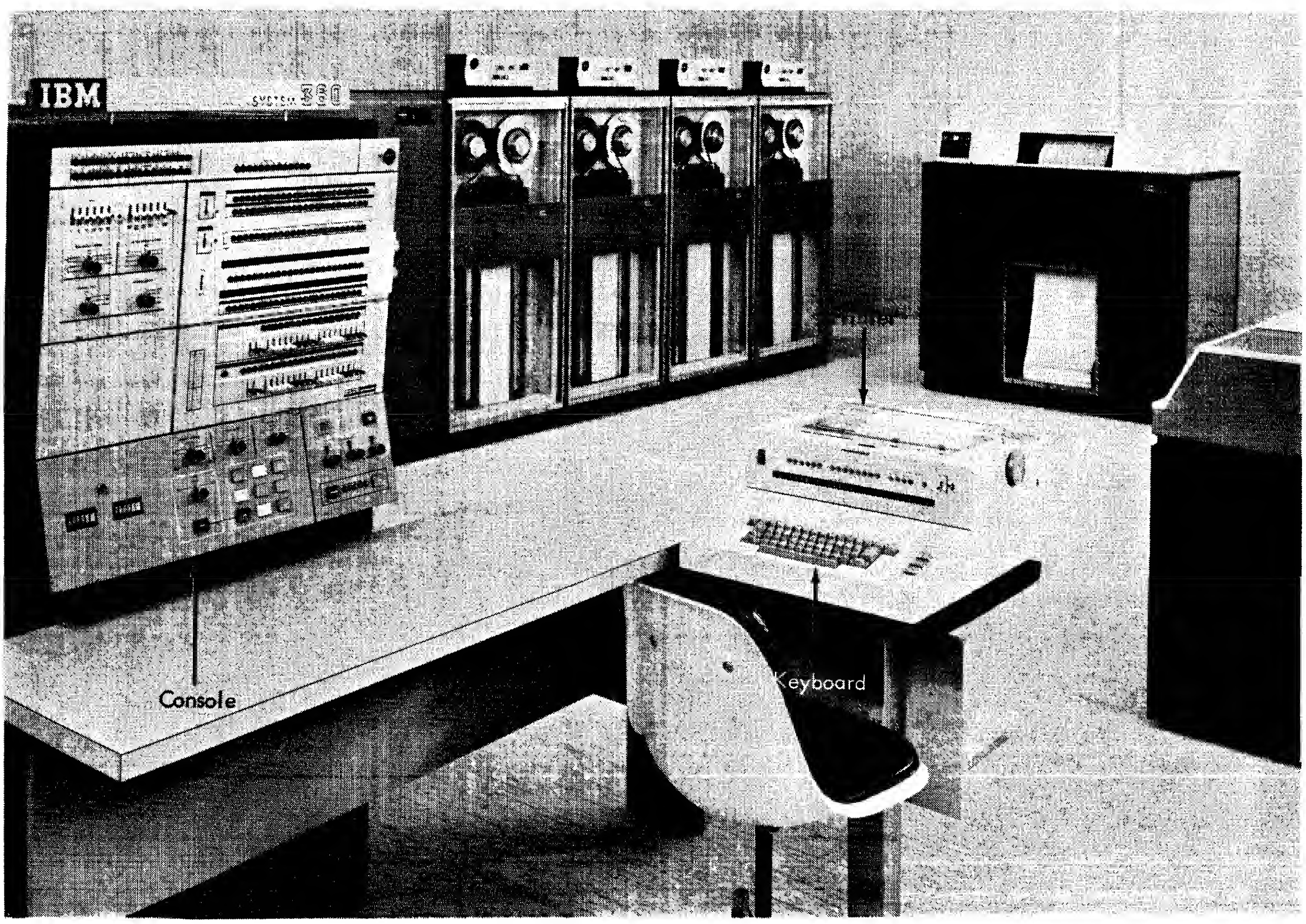


Figure 12. IBM System/360 Model 40 Console, Console Printer, and Keyboard

Symbols convey information; the symbol itself is not the information but merely represents it. The printed characters on this page are symbols that convey one meaning to some persons, a different meaning to others, and no meaning to those who do not know their significance (Figure 14).

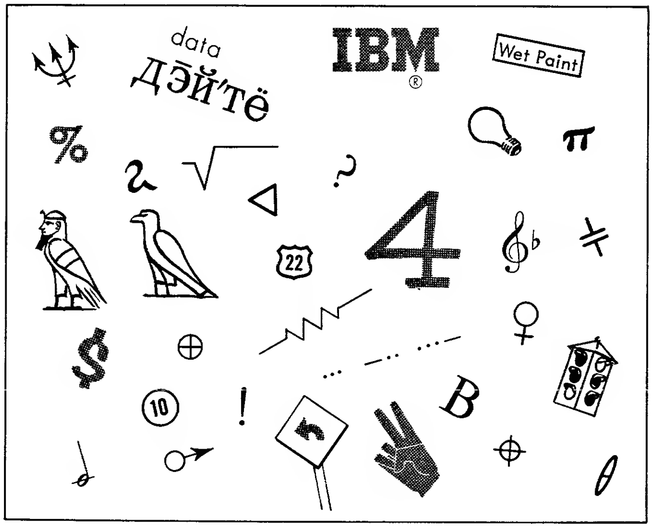


Figure 14. Symbols for communication

Presenting data to the computer system is similar in many ways to communicating with another person by letter. The intelligence to be conveyed must be reduced to a set of symbols. In the English language, these are the familiar letters of the alphabet, numbers and punctuation. The symbols are recorded on paper in a prescribed sequence and transported to another person who reads and interprets them.

Similarly, communication with the computer system requires that data be reduced to a set of symbols that can be read and interpreted by data processing machines. The symbols differ from those commonly used by people, because the information to be represented must conform to the design and operation of the machine. The choice of these symbols (and their meaning) is a matter of convention on the part of the designers. The important fact is that information can be represented by symbols, which become a language for the communication between people and machines.

Information to be used with the computer systems can be in the form of punched cards, paper tape, magnetic tape, direct access storage devices (DASD), magnetic ink characters, optically recognizable characters, microfilm and display screen images, communication network signals, etc. The list is growing larger each year. Some are pictured in Figure 15.

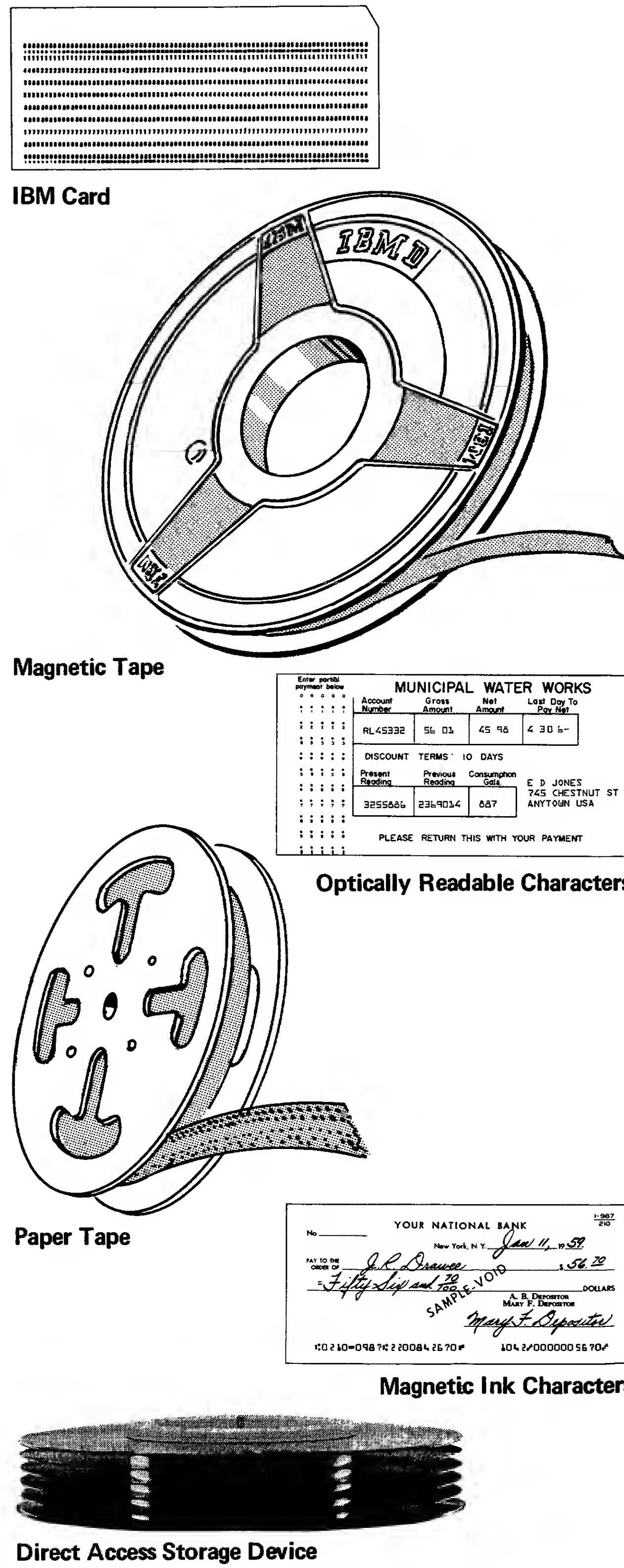


Figure 15. Data recording media

Data is represented on the punched card by the presence of small rectangular holes in specific locations of the card. In a similar manner, small circular holes along a paper tape represent data. On magnetic tape, or DASD, the symbols are small magnetized areas, called spots or bits, arranged in specific patterns. Magnetic ink characters are printed on paper. The shape of the characters and the magnetic properties of the ink permit the printed data to be read by both man and machine. The shape of the optical characters, together with the contrast with the background paper, permits optical characters to be read by the machine and by people.

Each medium requires a code or specific arrangement of symbols to represent data. These codes are described later in this section.

An input device of the computer system is a machine designed to sense or read information from one of the recording media. In the reading process, recorded data is converted to, or symbolized in, electronic form; the data can then be used by the machine for data processing operations.

An output device is a machine that receives information from the computer system and records it on the designated output medium.

All input/output devices cannot be used directly with all computer systems. However, data recorded on one medium can be transcribed to another medium for use with a different system. For example, data on cards or paper tape can be transcribed onto magnetic tape. Conversely, data on magnetic tape can be converted to cards, paper, tape, printed reports, or plotted graphs.

As there is communication between people and machines, there is also communication from one machine to another (Figure 16). This intercommunication may be the direct exchange of data (in electronic form) over wires, cables, or radio waves, or recorded output of one machine or system may be used as input to another machine or system.

Computer Data Representation

Not only must there be a method of representing data on cards, on paper tape, on magnetic tape, and in magnetic ink characters, but there must also be a method of representing data within a machine.

In the computer, data is represented by many electronic components: transistors, magnetic cores, wires, and so on. The storage and flow of data through these devices are represented as electronic signals or indications. The presence or absence of these signals in specific circuitry is the method of representing data, much as the presence of holes in a card represents data.

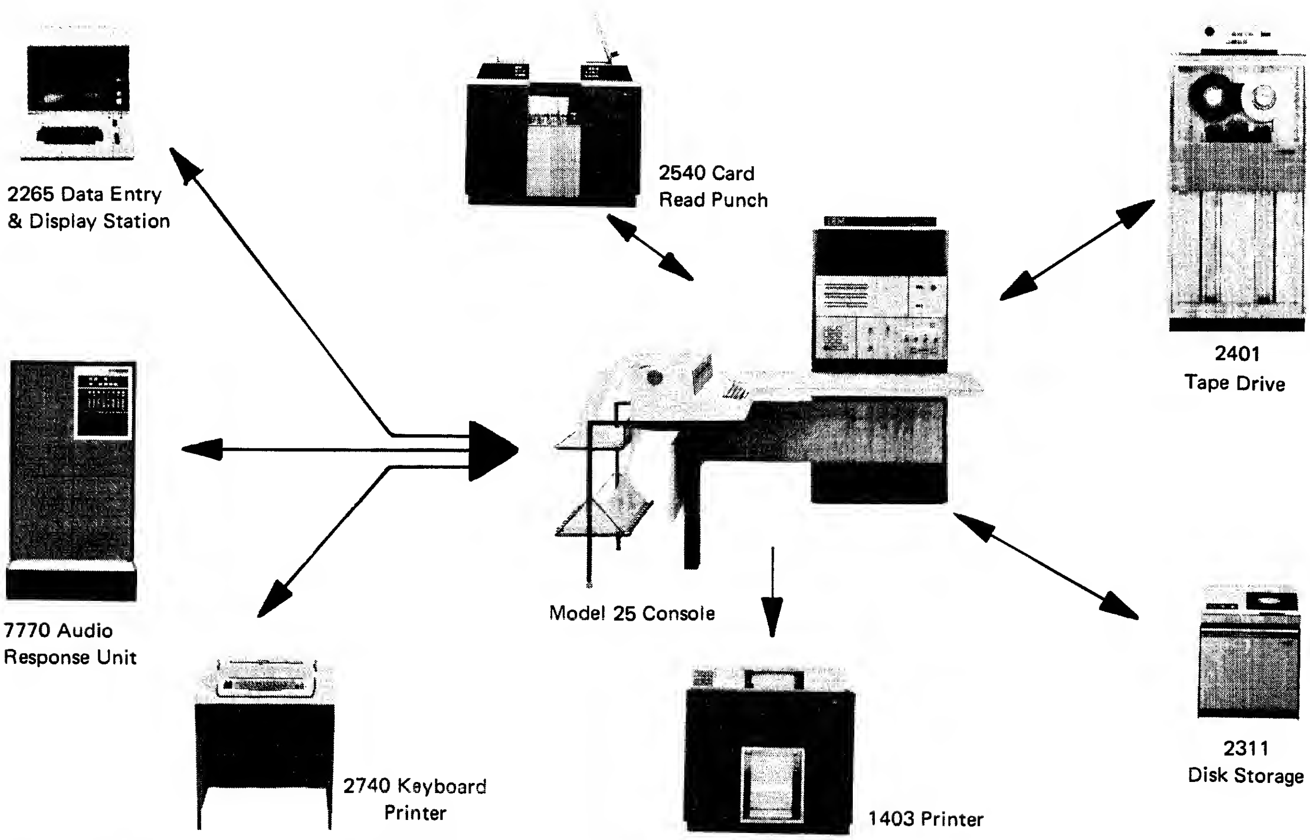


Figure 16. Machine-to-machine communication

Binary States

Computers function in binary states; this means that the computer components can indicate only two possible states or conditions. For example, the ordinary light bulb operates in a binary mode, that is, it is either on or off. Likewise, within the computer, transistors are maintained either conducting or nonconducting; magnetic materials are magnetized in one direction or in an opposite direction; and specific voltage potentials are present or absent (Figure 17). The binary states of operation of the components are signals to the computer, as the presence or absence of light from an electric light bulb can be a signal to a person.

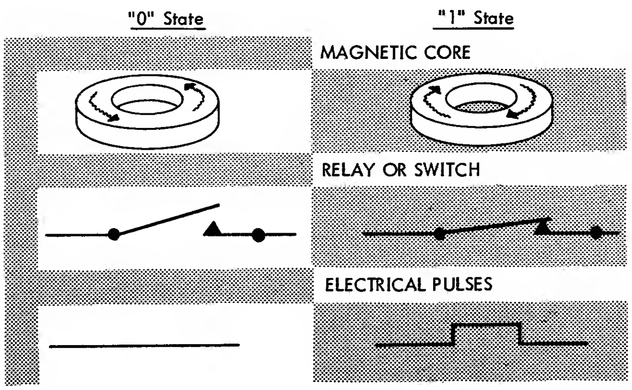


Figure 17. Binary components

Representing data within the computer is accomplished by assigning or associating a specific value to a binary indication or group of binary indications. For example, a device to represent decimal values could be designed with four electric light bulbs and switches to turn each bulb on or off (Figure 18).

The bulbs are assigned decimal values of 1, 2, 4, and 8. When a light is on, it represents the decimal value associated with it. When a light is off, the decimal value is not considered. With such an arrangement, the single decimal value represented by the four bulbs

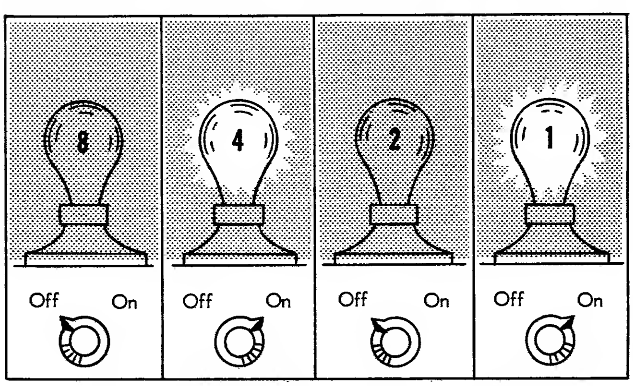


Figure 18. Representing decimal data with binary components

will be the numeric sum indicated by the lighted bulbs.

Decimal values 0 through 15 can be represented. The numeric value 0 is represented by all lights off; the value 15, by all lights on; 9, by having the 8 and 1 lights on and the 4 and 2 lights off; 5, by the 1 and 4 lights on and the 8 and 2 lights off; and so on.

The value assigned to each bulb or indicator in the example could have been something other than the values used. This change would involve assigning new values and determining a new scheme of operation. In a computer, the values assigned to a specific number of binary indications become the code or language for representing data.

Because binary indications represent data within a computer, a binary method of notation is used to illustrate these indications. The binary system of notation uses only two symbols, zero (0) and one (1), to represent specific values. In any one position of binary notation, the 0 represents the absence of a related or assigned value, and the 1 represents the presence of a related or assigned value. For example, to illustrate the indications of the light bulb in Figure 18, the following binary notation would be used: 0101.

The binary notations 0 and 1 are commonly called bits. Properly, they are called 0 bit and 1 bit. Occasionally, however, they are loosely spoken of as no bit (0 bit) and bit (1 bit). For example, the binary notation 0101 of Figure 18 would be described as having a 1 bit in the 1 and 4 bit positions, and a 0 bit in the 2 and 8 bit positions.

Binary Number Systems

In some computers, the values associated with the binary notation are related directly to the binary system. This system is not used in all computers, but the method of representing values using this numbering system is useful in learning the general concept of data representation.

The common decimal number system uses ten symbols or digits to represent numeric values, and the place value of the digits signifies units, tens, hundreds, thousands, and so on. The binary or base-two number system uses only two symbols or digits: 0 and 1. The position value of the bit symbols (0 or 1) is based on the progression of powers of 2; the units position of a binary number has the value of 1; the next position, a value of 2; the next, 4; the next, 8; the next, 16; and so on (Figure 19).

Figure 19. Place value of binary numbers

In pure binary notation, the binary digits or bits indicate whether the corresponding power of 2 is absent or present in each position of the number. The 1 bit represents the presence of the value, and the 0 bit represents the absence of the value. The place value of the digits does not signify units, tens, hundreds, or thousands, as in the decimal system; instead, the place value signifies units, twos, fours, eights, sixteens, and so on. Using this system the quantity 12, for example, is expressed with the symbols 1100, meaning $(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$, or $(1 \times 8) + (1 \times 4) + (0 \times 2) + (0 \times 1)$.

Figure 20 shows the binary representation of the decimal values 0 through 16.

Decimal Value	Place Value				
	16	8	4	2	1
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0

Figure 20. Binary representation of decimal values 0-16

Note that the decimal digits 0 through 9 are expressed by four binary digits. The system of coding or expressing decimal digits in an equivalent binary value is known as binary coded decimal (BCD). For example, the decimal value 265, 498 would appear in binary coded decimal form as shown in Figure 21.

Decimal Digits	2				6				5				4				9				8			
Binary Value	0	0	1	0	0	1	1	0	0	1	0	1	0	1	0	0	1	0	0	1	1	0	0	0
Place Value	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1

Figure 21. Binary coded decimal representation of decimal number 265, 498

Computer Codes

The method used to represent (symbolize) data is known as a code or a system. In the computer, the code relates data to a fixed number of binary indications (symbols). For example, a code used to represent numeric and alphabetic characters may use eight positions of binary indication. By the proper arrangement of the binary indications (0 bit, 1 bit), all characters can be represented by a different combination of bits.

Some computer codes in use are six-bit alphameric code, eight-bit alphameric code, two-out-of-five-count code, and six-bit (packed) numeric code.

Code Checking

Most computer codes are self-checking; that is, they are provided with a built-in method of checking the validity of the coded information. This code checking occurs automatically within the machine as the data processing operations are carried out. The method of validity checking is part of the design of the code.

In some codes, each unit or character of data is represented by a specific number of bit positions that must always contain an even number of 1 bits. Different characters are made up of different combinations of 1 bits, but the number of 1 bits in any valid character is always even. With this code system, a character with an odd number of 1 bits is detected, and an error is indicated. Likewise, a code may be used in which all characters must have an odd number of 1 bits; an error is indicated when characters with an even number of 1 bits are detected.

This type of checking is known as a parity check. Codes that use an even number of 1 bits are said to have even parity. Codes that use an odd number of bits are said to have odd parity.

In other codes, the number of 1 bits present in each unit of data is fixed. For example, a code may use eight bit positions to code all characters, but exactly four 1 bits will be present in each character. Characters having more or fewer than four 1 bits cause an error indication. This system of checking is known as a fixed-count check and is often used for data transmission in teleprocessing networks.

Six-Bit Alphameric Code (Binary Coded Decimal System)

In this code, all characters — numeric, alphabetic, and special — are represented (coded) using six positions of binary notation (plus a parity bit position). These positions are divided into three groups: one check position, two zone positions, and four numeric positions (Figure 22).

Check Bit	Zone Bits		Numeric Bits			
C	B	A	8	4	2	1

Figure 22. Bit positions, six-bit alphameric code

The four numeric positions are assigned decimal values of 8, 4, 2, and 1, and represent, in binary coded decimal form, the numeric digits 0 through 9 (Figure 23). Note that 0 is represented as 1010, actually the binary number for 10. The B and A zone bits are not present (00) when the numeric digits 0 through 9 are represented.

Decimal Digit	Place Value 8	4	2	1
0	1	0	1	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Figure 23. Numeric bit configurations, decimal digits 0-9, six-bit alphameric code

Combinations of zone and numeric bits represent alphabetic and special characters. The B and A bits provide for four possible bit combinations: 10, 01, 11, and 00.

The C position, known as the check bit, is used for code checking only. Because the six-bit alphameric code is usually an even parity code, the total number of bits used to represent a character must have an even number of 1 bits, or the character is considered invalid. A 1 bit is added to a character when the sum of the zone and numeric bits used to represent the character is odd. If the number of bits in a character is even without the C bit, the C bit is a 0.

Standard BCD Interchange Code

To provide compatibility of data for interchangeability among various computer systems, the standard BCD interchange code has been developed. This coding structure consists basically of 64 different characters.

Figure 24 shows the BCD standard interchange code used on the IBM 1401, 1410, 7010, 7040, and 7044 Data Processing Systems.

Shown are the collating sequence, graphics, card code, and BCD code for each of the 64 different bit combinations. The C bit, used for parity checking purposes, is dependent upon the specific computer design using the standard BCD interchange code. If the system uses odd parity, a C bit will be automatically placed in each C position of a character that contains an even number of bits. Conversely, a system using even parity will have a C bit placed in each C position of a character that contains an odd number of bits.

Collating Sequence	CHARACTER Report	Program	CARD CODE	BCD CODE (Core Storage)						
Low→	b		No Punches	C						
	.		12-3-8		B	A	8		2	1
	□	)	12-4-8	C	B	A	8	4		
	[		12-5-8		B	A	8	4		1
	<		12-6-8		B	A	8	4	2	
	⧻		12-7-8	C	B	A	8	4	2	1
	&	+	12	C	B	A				
	$		11-3-8	C	B		8		2	1
	*		11-4-8		B		8	4		
	]		11-5-8	C	B		8	4		1
	;		11-6-8	C	B		8	4	2	
	Δ		11-7-8		B		8	4	2	1
	–		11		B					
	/		0-1	C		A				1
	,		0-3-8	C		A	8		2	1
	%	(	0-4-8			A	8	4		
	⌒		0-5-8	C		A	8	4		1
	\		0-6-8	C		A	8	4	2	
	⧺		0-7-8			A	8	4	2	1
	b̸		2-8			A				
	#	=	3-8				8		2	1
	@	'	4-8	C			8	4		
	:		5-8				8	4		1
	>		6-8				8	4	2	
	√		7-8	C			8	4	2	1
	?		12-0	C	B	A	8		2	
COLLATING SEQUENCE	A		12-1		B	A				1
	B		12-2		B	A			2	
	C		12-3	C	B	A			2	1
	D		12-4		B	A		4		
	E		12-5	C	B	A		4		1
	F		12-6	C	B	A		4	2	
	G		12-7		B	A		4	2	1
	H		12-8		B	A	8			
	I		12-9	C	B	A	8			1
	!		11-0		B		8		2	
	J		11-1	C	B					1
	K		11-2	C	B				2	
	L		11-3		B				2	1
	M		11-4	C	B			4		
	N		11-5		B			4		1
	O		11-6		B			4	2	
	P		11-7	C	B			4	2	1
	Q		11-8	C	B		8			
	R		11-9		B		8			1
	⧧		0-2-8			A	8		2	
	S		0-2	C		A			2	
	T		0-3			A			2	1
	U		0-4	C		A		4		
	V		0-5			A		4		1
	W		0-6			A		4	2	
	X		0-7	C		A		4	2	1
	Y		0-8	C		A	8			
	Ƶ		0-9			A	8			1
	Ø		0	C			8		2	
	1		1							1
	2		2						2	
	3		3	C					2	1
	4		4					4		
	5		5	C				4		1
	6		6	C				4	2	
	7		7					4	2	1
	8		8				8			
High→	9		9	C			8			1

NOTE: Tape may use even parity.

Figure 24. Standard BCD interchange code

Five of the standard BCD bit combinations print out as two different characters (called graphics), depending upon the type set used in the printer. The two variations are called graphic subset 1 and graphic subset 2 (Figure 25).

BCD Code	Graphic Subset 1 Print Arrangement A	Graphic Subset 2 Print Arrangement H
8-2-1	#	=
8-4	@	'
A-8-4	%	(
B-A	&	+
B-A-8-4	⌑	)

Figure 25. Graphic subsets 1 and 2

Graphic subset 1 is used primarily for computer report writing and most commercial uses, while graphic subset 2 is used for advanced programming languages, such as FORTRAN and COBOL, and meets general requirements for mathematical symbolism.

Figure 26 indicates the preferred standardized terminology for the special characters included in the standard BCD interchange code.

SYMBOL	NAME
≢	Group Mark
‡	Record Mark
⧻	Segment Mark
ᵐ	Word Separator
@	At Sign
#	Number Sign
&	Ampersand
+	Plus
*	Asterisk
%	Percent
/	Slash
\	Backslash
⌑	Lozenge
b	Blank
ƀ	Substitute Blank
(	Left Parenthesis
)	Right Parenthesis
[	Left Bracket
]	Right Bracket
√	Tape Mark
<	Less than
>	Greater than
=	Equal to
;	Semicolon
:	Colon
.	Period or Point
'	Prime or Apostrophe
-	Minus or Hyphen (Dash)
Δ	Delta

Figure 26. Special character names

Eight-bit Alphameric Code (Extended Binary Coded Decimal Interchange Code — EBCDIC)

This code (Figure 27) uses eight binary positions for each character format, plus a position for parity checking. By using eight bit positions, 256 different characters can be coded. This code permits, for instance, the coding of uppercase and lowercase alphabetic characters, a much wider range of special characters, and many control characters that are meaningful to certain input/output devices. At present, many bit patterns have no assigned function (control or graphic). They are reserved for future assignment. EBCDIC is one of the two principal coding schemes for System/360.

Eight-Bit Alphameric Code (USASCII-8)

The USA Standard Code for Information Interchange (USASCII) is a seven-bit code developed through the cooperation of users of equipment of communications and data processing industries, in an attempt to simplify and standardize machine-to-machine and system-to-system communication.

Because the System/360 has an eight-bit character capacity, it was necessary to expand USASCII to an eight-bit representation. This expanded representation is referred to by IBM as USASCII-8. This code may be used for internal processing and input/output purposes with System/360 in those media for which USASCII has been standardized.

Computer Number Systems and Conversion

Binary System

Computers using the binary system of data representation are typified by the IBM System/360.

For these systems, the basic unit of information is a byte. Four bytes constitute a word consisting of 32 consecutive bit positions of information which are interpreted as a unit, much as a character or a digit in other systems.

The bit sections within the word have a place significance related to the binary number system. That is, the place position of a bit in the word determines the value of the bit. In the binary number system, the decimal values of the places (from right to left) are 1, 2, 4, 8, 16, 32, 64, and so on as shown in Figure 19.

Although the place values of the bits of a word are always those of the binary number system, they can be interpreted or processed in such a way as to represent other than a binary number. For example, a 32-

EBCDIC	Bit Configuration	EBCDIC	Bit Configuration	EBCDIC	Bit Configuration
NUL	0000 0000		0100 0101		1000 1010
SOH	0000 0001		0100 0110		1000 1011
STX	0000 0010		0100 0111		1000 1100
ETX	0000 0011		0100 1000		1000 1101
PF	0000 0100		0100 1001		1000 1110
HT	0000 0101	¢ [	0100 1010		1000 1111
LC	0000 0110	.	0100 1011		1001 0000
DEL	0000 0111	<	0100 1100	j	1001 0001
	0000 1000	(	0100 1101	k	1001 0010
RLF	0000 1001	+	0100 1110	l	1001 0011
SMM	0000 1010	|	0100 1111	m	1001 0100
VT	0000 1011	&	0101 0000	n	1001 0101
FF	0000 1100		0101 0001	o	1001 0110
CR	0000 1101		0101 0010	p	1001 0111
SO	0000 1110		0101 0011	q	1001 1000
SI	0000 1111		0101 0100	r	1001 1001
DLE	0001 0000		0101 0101		1001 1010
DC1	0001 0001		0101 0110		1001 1011
DC2	0001 0010		0101 0111		1001 1100
TM	0001 0011		0101 1000		1001 1101
RES	0001 0100		0101 1001		1001 1110
NL	0001 0101	!]	0101 1010		1001 1111
BS	0001 0110	$	0101 1011		1010 0000
IL	0001 0111	*	0101 1100	—	1010 0001
CAN	0001 1000	)	0101 1101	s	1010 0010
EM	0001 1001	;	0101 1110	t	1010 0011
CC	0001 1010	¬	0101 1111	u	1010 0100
CU1	0001 1011	–	0110 0000	v	1010 0101
IFS	0001 1100	/	0110 0001	w	1010 0110
IGS	0001 1101		0110 0010	x	1010 0111
IRS	0001 1110		0110 0011	y	1010 1000
IUS	0001 1111		0110 0100	z	1010 1001
DS	0010 0000		0110 0101		1010 1010
SOS	0010 0001		0110 0110		1010 1011
FS	0010 0010		0110 0111		1010 1100
	0010 0011		0110 1000		1010 1101
BYP	0010 0100		0110 1001		1010 1110
LF	0010 0101	7/12	0110 1010		1010 1111
ETB	0010 0110	,	0110 1011		1011 0000
ESC	0010 0111	%	0110 1100		1011 0001
	0010 1000	_	0110 1101		1011 0010
	0010 1001	>	0110 1110		1011 0011
SM	0010 1010	?	0110 1111		1011 0100
CU2	0010 1011		0111 0000		1011 0101
	0010 1100		0111 0001		1011 0110
ENQ	0010 1101		0111 0010		1011 0111
ACK	0010 1110		0111 0011		1011 1000
BEL	0010 1111		0111 0100		1011 1001
	0011 0000		0111 0101		1011 1010
	0011 0001		0111 0110		1011 1011
SYN	0011 0010		0111 0111		1011 1100
	0011 0011		0111 1000		1011 1101
PN	0011 0100	6/0	0111 1001		1011 1110
RS	0011 0101	:	0111 1010		1011 1111
UC	0011 0110	#	0111 1011	PZ 7/11	1100 0000
EOT	0011 0111	@	0111 1100	A	1100 0001
	0011 1000	'	0111 1101	B	1100 0010
	0011 1001	=	0111 1110	C	1100 0011
	0011 1010	"	0111 1111	D	1100 0100
CU3	0011 1011		1000 0000	E	1100 0101
DC4	0011 1100	a	1000 0001	F	1100 0110
NAK	0011 1101	b	1000 0010	G	1100 0111
	0011 1110	c	1000 0011	H	1100 1000
SUB	0011 1111	d	1000 0100	I	1100 1001
SP	0100 0000	e	1000 0101		1100 1010
	0100 0001	f	1000 0110		1100 1011
	0100 0010	g	1000 0111	⑁	1100 1100
	0100 0011	h	1000 1000		1100 1101
	0100 0100	i	1000 1001	⑂	1100 1110

Figure 27. Configurations, Extended Binary Coded Decimal Interchange Code (EBCDIC)

EBCDIC	Bit Configuration
	1100 1111
MZ 7/13	1101 0000
J	1101 0001
K	1101 0010
L	1101 0011
M	1101 0100
N	1101 0101
O	1101 0110
P	1101 0111
Q	1101 1000
R	1101 1001
	1101 1010
	1101 1011
	1101 1100
	1101 1101
	1101 1110
	1101 1111
RM 5/12	1110 0000
	1110 0001
S	1110 0010
T	1110 0011
U	1110 0100
V	1110 0101
W	1110 0110
X	1110 0111
Y	1110 1000
Z	1110 1001
	1110 1010
	1110 1011
⑁	1110 1100
	1110 1101
	1110 1110
	1110 1111
0	1111 0000
1	1111 0001
2	1111 0010
3	1111 0011
4	1111 0100
5	1111 0101
6	1111 0110
7	1111 0111
8	1111 1000
9	1111 1001
⌇	1111 1010
	1111 1011
	1111 1100
	1111 1101
	1111 1110
EO	1111 1111

Figure 27. (Continued)

bit word (Figure 28) can be interpreted as one 32-place binary number, as an eight-digit hexadecimal number, as four alphameric characters (which may be alphabetic or numeric), or as any predetermined representation established by the programmer.

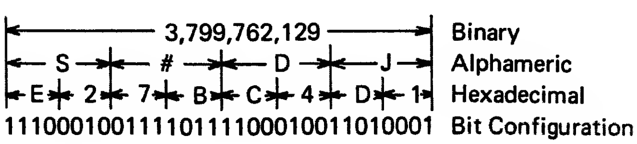


Figure 28. The 32-bit word

Octal System

It is apparent that binary numbers require several times as many positions as decimal numbers to display the equivalent number. In talking and writing, these binary numbers are bulky. A long string of ones and zeros cannot be effectively transmitted from one individual to another. Some shorthand method is necessary. The octal and hexadecimal number systems fill this need. Because of their simple relationship to binary, numbers can be converted from one system to another by inspection. The base or radix of the octal system is 8. This means there are eight symbols: 0, 1, 2, 3, 4, 5, 6, and 7. There are no 8s or 9s in this number system. The important relationship to remember is that three binary positions are equivalent to one octal position. The sample chart shown as Figure 29 is used to convert between the binary, octal, and decimal systems.

BINARY	OCTAL	DECIMAL
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

At this point, a carry to the next-higher position of the number is necessary, since all eight symbols have been used.

BINARY	OCTAL	DECIMAL
001 000	10	8
001 001	11	9
001 010	12	10
001 011	13	11
001 100	14	12
•	•	•
•	•	•
•	•	•

Figure 29. Binary octal decimal conversion

Remember that the internal circuitry of the computer understands only binary ones and zeros. The octal system is used to provide a shorthand method of reading and writing binary numbers. In the octal system, the base number is 8. The digits of the number represent the coefficients of the ascending powers of 8. Consider the octal number:

$$
\begin{aligned}
173 &= (1 \times 8^2) + (7 \times 8^1) + (3 \times 8^0) \\
&= \quad 64 \quad + \quad 56 \quad + \quad 3 \\
&= \quad 123 \text{ (decimal)}
\end{aligned}
$$

Similarly:

Octal 173

3 units	=	3
56 eights	=	56
64 sixty-fours	=	64

By remembering what a number represents in the binary or octal system, the number can be converted to its decimal equivalent by the method shown above. As the numbers get bigger, this method becomes more difficult to use. The following section provides detailed methods for converting from one system to another.

Integer Conversion

Decimal to Octal

Rule: Continuously divide the decimal number by 8, and develop the octal number from the remainders of each step of the division.

Example: Convert the decimal number 149 to its octal equivalent.

8 |149	Remainder 5	
8 |18	" 2	= 225
8 |2	" 2	
0		Read ↑

We first divide the original number to be converted by 8. The remainder of this division becomes the low-order digit of the conversion (5). We then divide the quotient (received from the first division) by 8. Again the remainder becomes a part of the answer (next-higher order, 2). This is continued until the quotient is smaller than the divisor. At this time the final quotient is considered the high order of the conversion (2).

Octal to Decimal

Example: Convert the octal number 225 to its decimal equivalent.

Rule: Continuously multiply by 8, and add the next octal digit.

$$
\begin{array}{r}
2\quad 2\quad 5 \\
\times\ 8 \\ \hline
16 \\
+\ 2 \\ \hline
18 \\
\times\ 8 \\ \hline
144 \\
+\ 5 \\ \hline
149
\end{array}
$$

The high-order digit is multiplied by 8, and the next-lower-order digit is added to the result. The resultant answer is then multiplied by 8, and the next-lower-order digit is added to the result. When the low-order digit has been added to the answer, the process ends. In the following examples, where multiplication or division is used, detailed explanations will not be used because the proceedings are similar.

Octal to Binary and Binary to Octal

Rule: Express the number in binary groups of three.

Examples:

Octal to Binary	Binary to Octal
2 2 5	010 010 101
010 010 101 = 010 010 101	2 2 5 = 225

Decimal to Binary

Rule: Divide the decimal number by 2, and develop the binary number from the remainders.

Example: Convert 149 to its binary equivalent.

2 |149	Remainder 1	
2 |74	" 0	
2 |37	" 1	
2 |18	" 0	
2 |9	" 1	= 010 010 101
2 |4	" 0	
2 |2	" 0	
2 |1	" 1	
0	"	Read ↑

Binary to Decimal

Rule: Continuously multiply by 2, and add the next binary digit.

Example: Convert 010 010 101 to its decimal equivalent.

$$
\begin{array}{r}
10\ 010\ 101 \\
\times\ 2 \\ \hline
2 \\
+\ 0 \\ \hline
2 \\
\times\ 2 \\ \hline
4 \\
+\ 0 \\ \hline
4 \\
\times\ 2 \\ \hline
8 \\
+\ 1 \\ \hline
9 \\
\times\ 2 \\ \hline
18 \\
+\ 0 \\ \hline
18 \\
\times\ 2 \\ \hline
36 \\
+\ 1 \\ \hline
37 \\
\times\ 2 \\ \hline
74 \\
+\ 0 \\ \hline
74 \\
\times\ 2 \\ \hline
148 \\
+\ 1 \\ \hline
149
\end{array}
$$

OR 10 010 101

$$= 1\ (2^7) + 0\ (2^6) + 0(2^5) + 1\ (2^4) + 0\ (2^3) + 1\ (2^2) + 0\ (2^1) + 1\ (2^0)$$

$$= 128 + 16 + 4 + 1$$

$$= 149$$

Fraction Conversion

Decimal to Octal

Rule: Multiply by 8, and develop the octal number from the carry.

Example:

```
Read     .149
   •    × 8
       ─────
     1  .192
        × 8
       ─────
     1  .536
        × 8
       ─────
     4  .288
        × 8
       ─────
     2  .304
     =  .1142 +
```

Octal to Decimal

Rule: Express as powers of 8, and add.

Example:

$.1142 = 1\ (8^{-1}) + 1\ (8^{-2}) + 4\ (8^{-3}) + 2\ (8^{-4})$

$= 1/8 + 1/64 + 4/512 + 2/4096$

$= 610/4096$

$= .1489$

or .149

Octal to Binary and Binary to Octal

Rule: The same rule applies for fractions as for whole numbers.

Example:

```
.1    1    4    2        .001  001  100  010
.001  001  100  010      .1    1    4    2
```

Binary to Decimal

Rule: The same rule applies as for whole numbers.

Example:

.001 001 100 010

$= 1\,(2^{-3}) + 1\,(2^{-6}) + 1\,(2^{-7}) + 1\,(2^{-11})$

$= 1/8 + 1/64 + 1/128 + 1/2048$

$= 305/2048$

$= .1489$ plus

or .149

Hexadecimal System

Whereas all IBM computers use a binary system to indicate the presence of a power of 2 up to the size of a full word or designated field, the basic man-machine communication with System/360 is in the hexadecimal numbering system. For example, assembler programs usually list the contents of storage in hexadecimal notation, and the literature describing operation codes and storage formats gives this information in hexadecimal notation.

Hexadecimal means 16. The hexadecimal system uses binary bits to count up to 16, carry, and then start counting again. It does not, however, count to 16 in numbers. It counts from 0 through 9 in numbers; then 10 is A, 11 is B, 12 is C, 13 is D, 14 is E, and 15 is F, in hexadecimal representation. It takes four binary bit positions to count to F (15) in a computer. Figure 30 shows how it is done.

DECIMAL SYSTEM	HEXADECIMAL SYSTEM	BINARY SYSTEM 8 4 2 1 Bit values
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1
6	6	0 1 1 0
7	7	0 1 1 1
8	8	1 0 0 0
9	9	1 0 0 1
10	A	1 0 1 0
11	B	1 0 1 1
12	C	1 1 0 0
13	D	1 1 0 1
14	E	1 1 1 0
15	F	1 1 1 1

Figure 30. Relationship among decimal, hexadecimal, and binary systems

Since System/360 has eight bits (plus a parity bit) in each byte of its storage, each byte can be thought of as being two hexadecimal-system digits; for example:

Decimal	248
Binary	1111 1000
Hexadecimal	F 8

Remember that the hexadecimal system, like the octal system, is simply a shorthand notation used to express the binary bit patterns within a computer such as System/360. Thus, it is also related to the other code structures defined previously. In extended binary coded decimal interchange code (EBCDIC), this eight-bit character (called F8 in hexadecimal) would be an 8; in USASCII-8 code, it would be an x; it could have other meanings in other codes. But, regardless of the code meaning, it is expressed as F8 in the hexadecimal system.

Integer Translation, Hexadecimal to Decimal

Suppose we have a hexadecimal number such as A4B5. How de we convert it to a decimal-system number? First, think of the rightmost (low-order) position as position 1, the next position to the left as position 2,

	HEX	DEC	HEX	DEC	HEX		HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
	2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
	3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
	4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
	5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
	6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
	7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
	8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
	9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
	A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
	B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
	C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
	D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
	E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
	F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
Hexadecimal Positions	8		7		6		5		4		3		2		1	

Figure 31. Hexadecimal — decimal integer conversion table

the next position to the left as position 3, and the leftmost position as position 4. Looking at Figure 31, note that:

5 in hex (hexadecimal) position 1 equals	5
B in hex position 2 equals	176
4 in hex position 3 equals	1,024
A in hex position 4 equals	40,960
The sum is the decimal value of A4B5	(42,165)

Note: Without reference tables, use the same conversion method described for octal to decimal, substituting 16 for 8 as the multiplicand.

Integer Translation, Decimal to Hexadecimal

Reversing the procedure, to convert the decimal number 16,428 to hex, look up in Figure 31 the next-smaller number than 16,428. Note the hex equivalent and position number. Subtract the decimal value of that hex digit from 16,428, and look up the remainder in Figure 31. The process works as follows:

Find the hex equivalent of decimal	16,428
4 in hex position 4 equals	16,384
Remainder	44
0 in position 3 equals	0
Remainder	44
2 in position 2 equals	32
Remainder	12
C in position 1 equals	12

Therefore, 402C is the hex equivalent of 16,428.

Note: Without tables, use the same conversion method as for decimal to octal, substituting 16 for 8 as the divisor.

HEX	0123 DEC	HEX	4 5 6 7 DECIMAL	HEX	0 1 2 3 DECIMAL	HEX	4 5 6 7 DECIMAL EQUIVALENT
.0	.0000	.00	.0000 0000	.000	.0000 0000 0000	.0000	.0000 0000 0000 0000
.1	.0625	.01	.0039 0625	.001	.0002 4414 0625	.0001	.0000 1525 8789 0625
.2	.1250	.02	.0078 1250	.002	.0004 8828 1250	.0002	.0000 3051 7578 1250
.3	.1875	.03	.0117 1875	.003	.0007 3242 1875	.0003	.0000 4577 6367 1875
.4	.2500	.04	.0156 2500	.004	.0009 7656 2500	.0004	.0000 6103 5156 2500
.5	.3125	.05	.0195 3125	.005	.0012 2070 3125	.0005	.0000 7629 3945 3125
.6	.3750	.06	.0234 3750	.006	.0014 6484 3750	.0006	.0000 9155 2734 3750
.7	.4375	.07	.0273 4375	.007	.0017 0898 4375	.0007	.0001 0681 1523 4375
.8	.5000	.08	.0312 5000	.008	.0019 5312 5000	.0008	.0001 2207 0312 5000
.9	.5625	.09	.0351 5625	.009	.0021 9726 5625	.0009	.0001 3732 9101 5625
.A	.6250	.0A	.0390 6250	.00A	.0024 4140 6250	.000A	.0001 5258 7890 6250
.B	.6875	.0B	.0429 6875	.00B	.0026 8554 6875	.000B	.0001 6784 6679 6875
.C	.7500	.0C	.0468 7500	.00C	.0029 2968 7500	.000C	.0001 8310 5468 7500
.D	.8125	.0D	.0507 8125	.00D	.0031 7382 8125	.000D	.0001 9836 4257 8125
.E	.8750	.0E	.0546 8750	.00E	.0034 1796 8750	.000E	.0002 1362 3046 8750
.F	.9375	.0F	.0585 9375	.00F	.0036 6210 9375	.000F	.0002 2888 1835 9375
Hexadecimal Positions: 1		2		3		4	

Figure 32. Hexadecimal — decimal fraction conversion table

Fraction Translation, Hexadecimal to Decimal

To convert from hex fractions to decimal fractions, use Figure 32 to find the sum of the decimal equivalents for each position of the fraction.

Example: Convert .ABC hex to decimal.

.A hex in position 1 equals	.6250
.0B hex in position 2 equals	.0429 6875
.00C hex in position 3 equals	.0029 2968 7500
.ABC hex equals decimal	.6708 9843 7500

Note: Without tables, use the same method as for octal to decimal, substituting 16 for 8 as the base in calculating the value of each digit.

Fraction Translation, Decimal to Hexadecimal

To convert from decimal to hex, find the next-lower decimal value and its hex equivalent in Figure 32. Subtract the found decimal value from the desired decimal value, and use this to locate the next hex equivalent. Repeat this procedure for the number of positions required.

Example: Convert .13 decimal to hex.

Decimal number to convert	.1300	to Hex
Next-lower decimal number	.1250	equals .2
Remainder	.0050 0000	
Next-lower decimal number	.0039 0625	.01
Remainder	.0010 9375 000	
Next-lower decimal number	.0009 7656 2500	.004
Remainder	.0001 1718 7500	
Next-lower decimal number	.0001 0681 1523 4375	.0007
.13 decimal approximately equals		hex .2147

Note: In the absence of convenient tables, use the same method as for decimal to octal, substituting 16 for 8 as the multiplicand.

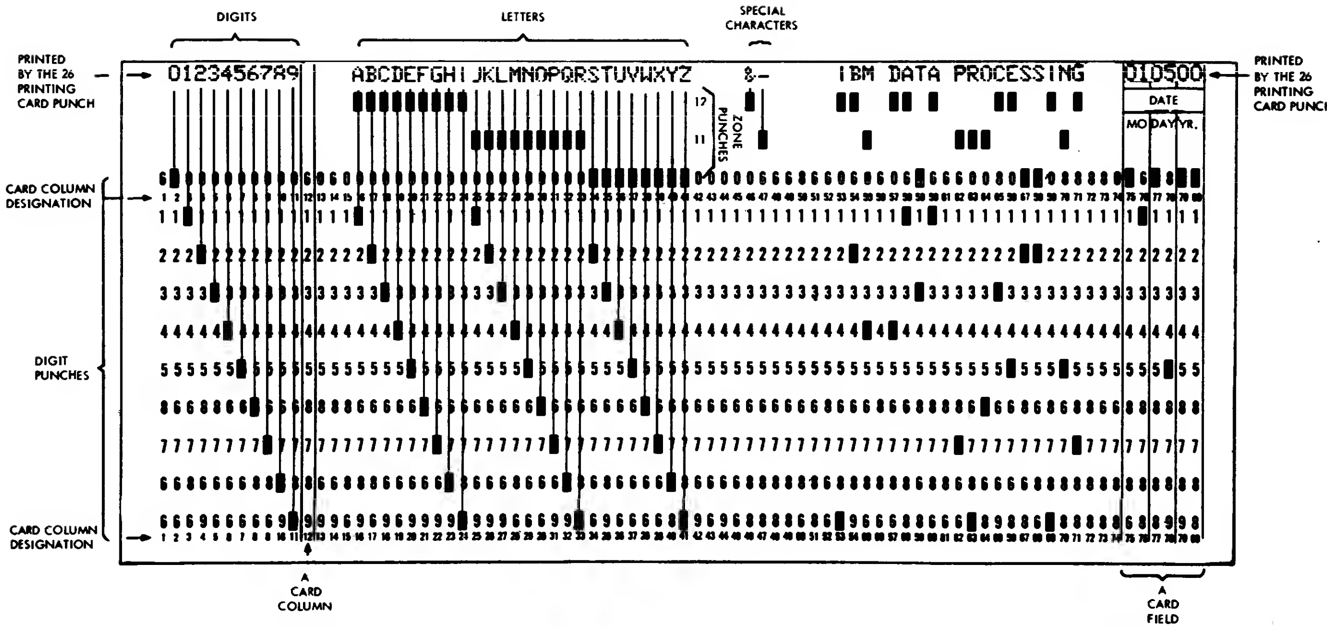


Figure 33. IBM punched card, standard hole pattern

Data Recording Media

IBM Cards

The IBM punched card is one of the most successful media for communication with machines. Information is recorded as small rectangular holes punched in specific locations in a standard size card (Figure 33). Information, represented (coded) by the presence or absence of holes in specific locations, can be read or sensed as the card is moved through a card-reading machine.

Reading or sensing the card is basically a process of automatically converting data, recorded as holes, to an electronic impulse and thereby entering the data into the machine. Cards are used both for entering the data into the machine and for recording or punching information from a machine. Thus, the card is not only a means of transferring data from some original source to a machine, but also is a common medium for the exchange of information between machines.

IBM's cards provide 80 vertical columns with twelve punching positions in each column. The twelve punching positions form twelve horizontal rows across the card. One or more punches in a single column represents a character. The number of columns used depends on the amount of data to be represented.

The card is often called a unit record, because the data is restricted to the 80 columns, and the card is read or punched as a unit of information. The actual data on the card, however, may consist of part of a record, one record, or more than one record. If more than 80 columns are needed to contain the data of a record, two or more cards may be used. Continuity between the cards of one record may be established by punching identifying information in designated columns of each card.

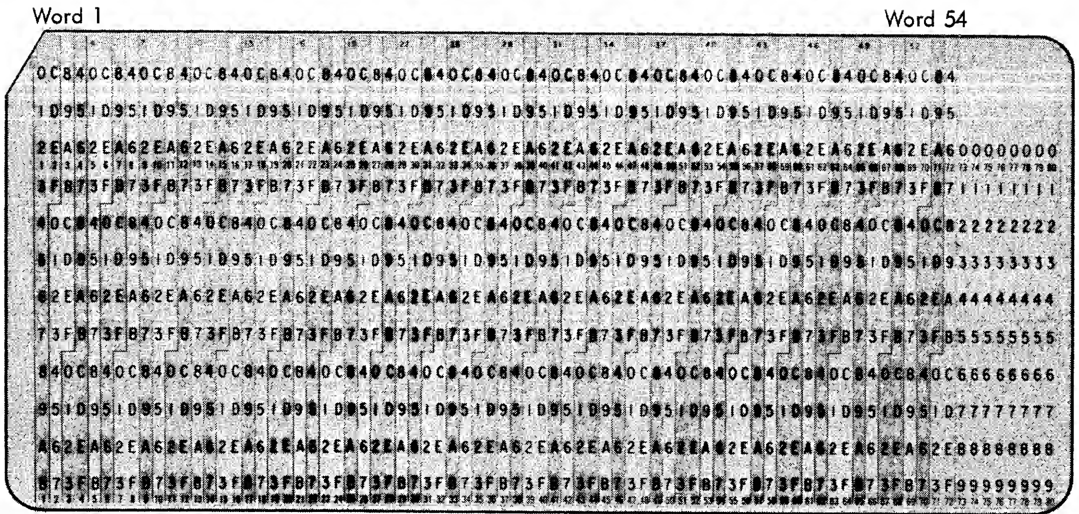
Information punched in cards is read or interpreted by a machine called a card reader and is recorded (punched) in a card by a machine called a card punch. Data is transcribed from source documents to punched cards by manually operated card punch machines.

IBM Card Code (Hole Pattern)

The standard IBM card code uses the twelve possible punching positions of a vertical column on a card to represent a numeric, alphabetic, or special character (Figure 33). The twelve hole positions are divided into two areas, numeric and zone. The first nine hole positions from the bottom edge of the card are the numeric hole positions and have an assigned value of 9, 8, 7, 6, 5, 4, 3, 2, and 1, respectively. The remaining three positions, 0, 11, and 12, are the zone positions. (The 0 position is considered to be both a numeric and a zone position.)

The numeric characters 0 through 9 are represented by a single hole in a vertical column. For example, 0 is represented by a single hole in the 0 zone position of the column.

The alphabetic characters are represented by two holes in a single vertical column, one numeric hole and one zone hole. The alphabetic characters A through I



Fifty-four words can be placed on a card (1-1/3 columns per word, four columns for three words).
The word numbers appear in every third column across the top of the card.

Figure 34. IBM 1130 binary card

use the twelve zone hole and a numeric hole 1 through 9, respectively. The alphabetic characters J through R use the 11 hole and a numeric hole 1 through 9, respectively. The alphabetic characters S through Z use the 0 zone hole and a numeric hole 2 through 9, respectively.

The standard special characters $ * % and so on, are represented by one, two, or three holes in a column of the card and consist of hole patterns not used to represent numeric or alphabetic characters.

Column Binary Data Representation

Column binary describes one method of recording binary information on cards. In this system, the information is arranged by words, starting in row 12 and going down through the column. Each punch represents a binary 1 in that position of the word. The IBM 1130 Computing System uses the column binary feature. Each word is made up of 16 bits and uses 1⅓ columns per word. Figure 34 is an example of an IBM 1130 column binary card.

Paper Tape

Punched paper tape serves much the same purpose as punched cards. Developed for transmitting telegraph messages over wires, paper tape is now used for data processing communication as well. For long-distance transmission, machines convert data from cards and keyboard strokes to paper tape, send the information over telephone or telegraph wires to produce a duplicate paper tape at the other end of the wire, and reconvert the information to punched cards, for later processing.

Data is recorded as a special arrangement of punched holes, precisely arranged along the length of a paper tape (Figures 35 and 36). Paper tape is a continuous recording medium, as compared to cards, which are fixed in length. Thus, paper tape can be used to record data in records of any length, limited only by the capacity of the storage medium into which the data is to be placed or from which the data is received.

Data punched in paper tape is read or interpreted by a paper tape reader and recorded by a paper tape punch.

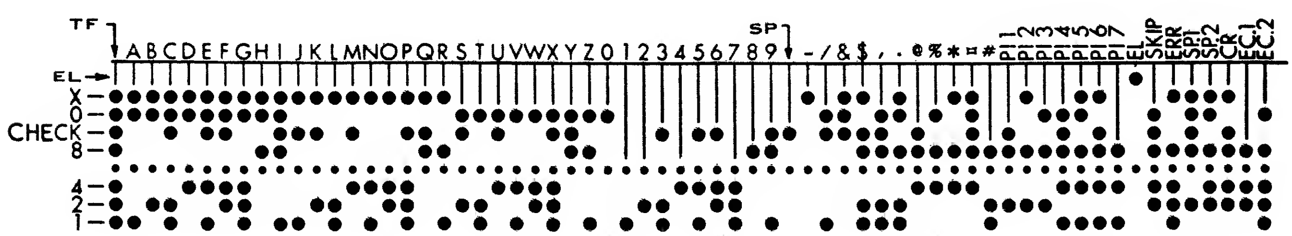


Figure 35. Paper tape, eight-channel code

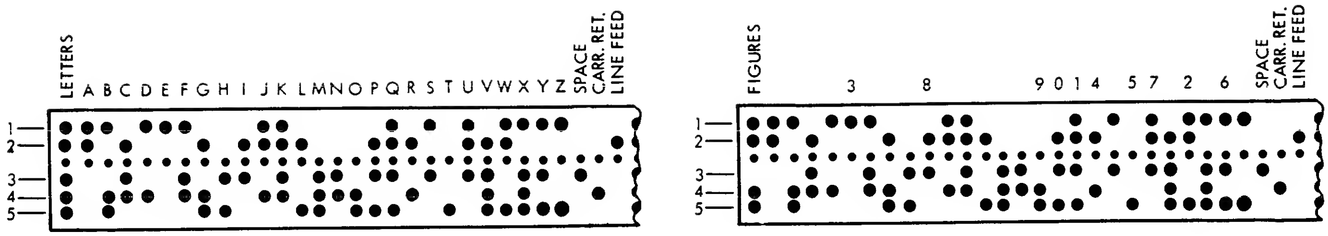


Figure 36. Paper tape, five-channel code

Eight-Channel Code (Hole Pattern)

Data is recorded (punched) and read as holes located in eight parallel channels along the length of the paper tape. One column of the eight possible punching positions (one for each channel) across the width of the tape is used to code numeric, alphabetic, special, and function characters. Figure 35 shows a section of paper tape with the eight channels and several coded characters.

The lower four channels of the tape (excluding the feed holes) are labeled 1, 2, 4, and 8 and are used to record numeric characters. The numeric characters 0 through 9 are represented as a punch or punches in these four positions. The sum of the position values indicates the numeric value of the character. For example, a hole in channel 1 is used to represent a numeric 1; a combination of a 1 and a 2 punch represents a numeric 3.

The X and 0 channels are similar to the zone punches in IBM punched cards. These channels are used in combination with the numeric channels to record alphabetic and special characters. The coding for the alphabetic and special characters is shown in Figure 35.

To check that each character is recorded correctly, each column of the tape is punched with an odd number of holes. A check hole must be present in the check channel for any column whose basic code (X, 0, 8, 4, 2, 1) consists of an even number of holes.

A punch in the EL (end-of-line) channel is a special function character used to mark the end of a record on the tape. The tape feed code consists of punches in the X, 0, 8, 4, 2, and 1 channels, and is used to indicate blank character positions. The paper tape reader automatically skips over the areas of tape punched with the tape feed code.

Five-Channel Code (Hole Pattern)

Data is recorded (punched) and read as holes in five parallel channels along the length of the paper tape. One column of the five possible punching positions (one of each channel) across the width of the tape is used to code numeric, alphabetic, special, and function characters. Figure 36 shows a section of paper tape with the five channels and several coded characters.

Because there are only 32 possible combinations of punches, using the five punching positions, a shift system is used to double the number of available codes. When the letters (LTRS) code punch precedes a section of tape, the characters that follow are interpreted as alphabetic characters (Figure 36). When the figures (FIGS) code punch precedes a section of tape, the coded punches are interpreted as numeric or special characters.

Ten of the 32 characters are used for coding both the alphabetic characters P, Q, W, E, R, T, Y, U, I, and O and the decimal digits 0 through 9, respectively. Interpretation depends on the shift code, LTRS or FIGS, preceding these characters. Likewise, the code for special characters is identical to that of other alphabetic characters. The actual alphabetic code that is equivalent to a given special character code varies, depending on customer requirements.

The function characters – space, carriage return (CR), and line feed (LF) – are the same in either LTRS or FIGS shift. The space code is used to indicate the absence of data on tape. The actual function of the CR and LF characters depends on the machine with which they are used.

Magnetic Tape

Magnetic tape is one of the principal input/output recording media for computer systems. It is also used extensively for storing intermediate results of computations and for compact storage of large files of data.

Magnetic tape units offer high-speed entry of data into the computer system, as well as efficient, extremely fast recording of processed data from the system. Highly reliable input/output data rates of up to 640,000 numeric characters per second are possible.

The magnetic tape unit functions as both an input unit and an output unit for the computer system. It moves the magnetic tape across a read/write head and accomplishes the actual reading and writing of information on the tape.

Information is recorded on magnetic tape as magnetized spots called bits. The recording can be retained indefinitely, or the recorded information can be automatically erased and the tape reused many times with continued high reliability.

So that tape can be easily handled and processed, it is wound on individual reels or in dust-resistant cartridges. Tape on the individual reels is ½ inch wide and is supplied in lengths of up to 2400 feet per reel. The cartridge-contained tape is threaded automatically.

The IBM 2420 magnetic tape unit model 7 reduces setup time with simplified automatic threading. In ten seconds, the free end of the tape is automatically threaded through the tape path from the supply reel to the takeup reel, positioned at the load point and loaded into the vacuum columns. Rewinding of a full 2,400-foot reel takes only one minute because the tape does not have to be removed from the vacuum columns.

Data is recorded in parallel channels or tracks along the length of the tape.

The tracks across the width of the tape provide one row of data. The spacing between the vertical rows is automatically generated during the writing operation and varies, depending on the character density used for recording. Character densities as high as 1600 characters per inch are available.

A longer space is automatically generated between blocks of records on the tape; this space is called the interblock gap (formerly called interrecord gap).

Data for IBM computers may be coded on seven-track magnetic tape in two modes, binary coded decimal (BCD) or binary. The code used depends on the computer that originates the data.

The IBM 2400 series tape units usually have a nine-track read/write head. A seven-track read/write head is available as an option. The nine-track heads read or write information in nine tracks along the length of the tape at a single density of 800 or 1600 bytes per inch, depending upon the model. Each byte consists of eight data bits and a parity bit. Each byte may represent two packed decimal digits, eight binary bits, or one special or alphameric character.

Packed mode is limited to recording decimal numbers, but it offers the advantage of recording two numbers in a single tape row, effectively doubling the reading and writing speed for numeric data on certain data processing systems.

BCD Coding on Tape

Letters of the alphabet, decimal numbers, and special characters may be recorded on magnetic tape using the binary coded decimal code (Figure 37).

Binary Coding on Tape

Some computer systems record data on magnetic tape in binary notation (Figure 38).

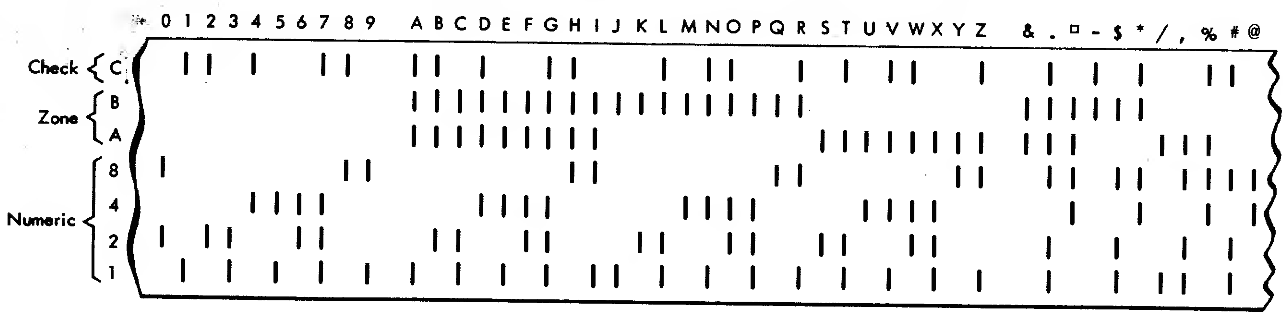


Figure 37. Magnetic tape, seven-track, seven-bit alphameric code

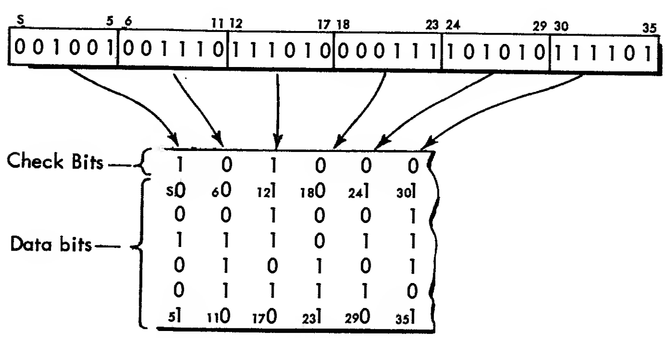


Figure 38. Magnetic tape, seven-track, binary recording

As in BCD recording, the C track is used to verify accuracy of tape reading and writing. With binary tape, however, each row of bits must contain an odd number of 1 bits.

The longitudinal parity check in binary mode is similar to that for BCD mode; the total number of bits in each horizontal track of a record block must be even.

Nine-Track Coding on Tape

The nine-track magnetic tape for Series 2400 tape units accepts the System/360 central processing unit coding, shown in Figure 39 as System/360 Eight-Bit Code. It adds an odd-parity bit, however, and the order of the bits in the character (byte) row is rearranged as shown in Figure 40.

Hexadecimal	Mnemonic	Graphic & Control Symbols		Punched Card Code	System/360 8-Bit Code
		BCDIC	EBCDIC		
00			NUL	12-0-9-8-1	0000 0000
01				12-9-1	0000 0001
02				12-9-2	0000 0010
03				12-9-3	0000 0011
04	SPM		PF	12-9-4	0000 0100
05	BALR		HT	12-9-5	0000 0101
06	BCTR		LC	12-9-6	0000 0110
07	BCR		DEL	12-9-7	0000 0111
08	SSK			12-9-8	0000 1000
09	ISK			12-9-8-1	0000 1001
0A	SVC			12-9-8-2	0000 1010
0B				12-9-8-3	0000 1011
0C	(EBCDIC +)			12-9-8-4	0000 1100
0D	(EBCDIC –)			12-9-8-5	0000 1101
0E				12-9-8-6	0000 1110
0F				12-9-8-7	0000 1111
10	LPR			12-11-9-8-1	0001 0000
11	LNR			11-9-1	0001 0001
12	LTR			11-9-2	0001 0010
13	LCR			11-9-3	0001 0011
14	NR		RES	11-9-4	0001 0100
15	CLR		NL	11-9-5	0001 0101
16	OR		BS	11-9-6	0001 0110
17	XR		IL	11-9-7	0001 0111
18	LR			11-9-8	0001 1000
19	CR			11-9-8-1	0001 1001
1A	AR			11-9-8-2	0001 1010
1B	SR			11-9-8-3	0001 1011
1C	MR			11-9-8-4	0001 1100
1D	DR			11-9-8-5	0001 1101
1E	ALR			11-9-8-6	0001 1110
1F	SLR			11-9-8-7	0001 1111
20	LPDR			11-0-9-8-1	0010 0000
21	LNDR			0-9-1	0010 0001
22	LTDR			0-9-2	0010 0010
23	LCDR			0-9-3	0010 0011
24	HDR		BYP	0-9-4	0010 0100
25			LF	0-9-5	0010 0101
26			EOB	0-9-6	0010 0110
27			PRE	0-9-7	0010 0111
28	LDR			0-9-8	0010 1000
29	CDR			0-9-8-1	0010 1001
2A	N ADR		SM	0-9-8-2	0010 1010
2B	N SDR			0-9-8-3	0010 1011
2C	N MDR			0-9-8-4	0010 1100
2D	N DDR			0-9-8-5	0010 1101
2E	AWR			0-9-8-6	0010 1110
2F	SWR			0-9-8-7	0010 1111
30	LPER			12-11-0-9-8-1	0011 0000
31	LNER			9-1	0011 0001
32	LTER			9-2	0011 0010
33	LCER			9-3	0011 0011
34	HER		PN	9-4	0011 0100
35			RS	9-5	0011 0101
36			UC	9-6	0011 0110
37			EOT	9-7	0011 0111
38	LER			9-8	0011 1000
39	CER			9-8-1	0011 1001
3A	N AER			9-8-2	0011 1010
3B	N SER			9-8-3	0011 1011
3C	N MER			9-8-4	0011 1100
3D	N DER			9-8-5	0011 1101
3E	AUR			9-8-6	0011 1110
3F	SUR			9-8-7	0011 1111
40	STH		SP	no punches	0100 0000
41	LA			12-0-9-1	0100 0001
42	STC			12-0-9-2	0100 0010
43	IC			12-0-9-3	0100 0011
44	EX			12-0-9-4	0100 0100
45	BAL			12-0-9-5	0100 0101
46	BCT			12-0-9-6	0100 0110
47	BC			12-0-9-7	0100 0111
48	LH			12-0-9-8	0100 1000
49	CH			12-8-1	0100 1001
4A	AH		¢	12-8-2	0100 1010
4B	SH	.	.	12-8-3	0100 1011
4C	MH	⌑)	<	12-8-4	0100 1100
4D		[	(	12-8-5	0100 1101
4E	CVD	<	+	12-8-6	0100 1110
4F	CVB	‡	|	12-8-7	0100 1111
50	ST	& +	&	12	0101 0000
51				12-11-9-1	0101 0001
52				12-11-9-2	0101 0010
53				12-11-9-3	0101 0011
54	N			12-11-9-4	0101 0100
55	CL			12-11-9-5	0101 0101

Hexadecimal	Mnemonic	Graphic & Control Symbols		Punched Card Code	System/360 8-Bit Code
		BCDIC	EBCDIC		
56	O			12-11-9-6	0101 0110
57	X			12-11-9-7	0101 0111
58	L			12-11-9-8	0101 1000
59	C			11-8-1	0101 1001
5A	A		!	11-8-2	0101 1010
5B	S	$	$	11-8-3	0101 1011
5C	M	*	*	11-8-4	0101 1100
5D	D	]	)	11-8-5	0101 1101
5E	AL	;	;	11-8-6	0101 1110
5F	SL	Δ	¬	11-8-7	0101 1111
60	STD	-	-	11	0110 0000
61		/	/	0-1	0110 0001
62				11-0-9-2	0110 0010
63				11-0-9-3	0110 0011
64				11-0-9-4	0110 0100
65				11-0-9-5	0110 0101
66				11-0-9-6	0110 0110
67				11-0-9-7	0110 0111
68	LD			11-0-9-8	0110 1000
69	CD			0-8-1	0110 1001
6A	N AD			12-11	0110 1010
6B	N SD	,	,	0-8-3	0110 1011
6C	N MD	% (	%	0-8-4	0110 1100
6D	N DD	∨	—	0-8-5	0110 1101
6E	AW	\	>	0-8-6	0110 1110
6F	SW	⧻	?	0-8-7	0110 1111
70	STE			12-11-0	0111 0000
71				12-11-0-9-1	0111 0001
72				12-11-0-9-2	0111 0010
73				12-11-0-9-3	0111 0011
74				12-11-0-9-4	0111 0100
75				12-11-0-9-5	0111 0101
76				12-11-0-9-6	0111 0110
77				12-11-0-9-7	0111 0111
78	LE			12-11-0-9-8	0111 1000
79	CE			8-1	0111 1001
7A	N AE	ƀ	:	8-2	0111 1010
7B	N SE	# =	#	8-3	0111 1011
7C	N ME	@'	@	8-4	0111 1100
7D	N DE	:	'	8-5	0111 1101
7E	AU	>	=	8-6	0111 1110
7F	SU	√	"	8-7	0111 1111
80	SSM			12-0-8-1	1000 0000
81			a	12-0-1	1000 0001
82	LPSW		b	12-0-2	1000 0010
83	(Diagnose)		c	12-0-3	1000 0011
84	WRD		d	12-0-4	1000 0100
85	RDD		e	12-0-5	1000 0101
86	BXH		f	12-0-6	1000 0110
87	BXLE		g	12-0-7	1000 0111
88	SRL		h	12-0-8	1000 1000
89	SLL		i	12-0-9	1001 1001
8A	SRA			12-0-8-2	1000 1010
8B	SLA			12-0-8-3	1000 1011
8C	SRDL			12-0-8-4	1000 1100
8D	SLDL			12-0-8-5	1000 1101
8E	SRDA			12-0-8-6	1000 1110
8F	SLDA			12-0-8-7	1000 1111
90	STM			12-11-8-1	1001 0000
91	TM		j	12-11-1	1001 0001
92	MVI		k	12-11-2	1001 0010
93	TS		l	12-11-3	1001 0011
94	NI		m	12-11-4	1001 0100
95	CLI		n	12-11-5	1001 0101
96	OI		o	12-11-6	1001 0110
97	XI		p	12-11-7	1001 0111
98	LM		q	12-11-8	1001 1000
99			r	12-11-9	1001 1001
9A				12-11-8-2	1001 1010
9B				12-11-8-3	1001 1011
9C	SIO			12-11-8-4	1001 1100
9D	TIO			12-11-8-5	1001 1101
9E	HIO			12-11-8-6	1001 1110
9F	TCH			12-11-8-7	1001 1111
A0				11-0-8-1	1010 0000
A1				11-0-1	1010 0001
A2			s	11-0-2	1010 0010
A3			t	11-0-3	1010 0011
A4			u	11-0-4	1010 0100
A5			v	11-0-5	1010 0101
A6			w	11-0-6	1010 0110
A7			x	11-0-7	1010 0111
A8			y	11-0-8	1010 1000
A9			z	11-0-9	1010 1001
AA				11-0-8-2	1010 1010

Hexadecimal	Mnemonic	Graphic & Control Symbols		Punched Card Code	System/360 8-Bit Code
		BCDIC	EBCDIC		
AB				11-0-8-3	1010 1011
AC				11-0-8-4	1010 1100
AD				11-0-8-5	1010 1101
AE				11-0-8-6	1010 1110
AF				11-0-8-7	1010 1111
B0				12-11-0-8-1	1011 0000
B1				12-11-0-1	1011 0001
B2				12-11-0-2	1011 0010
B3				12-11-0-3	1011 0011
B4				12-11-0-4	1011 0100
B5				12-11-0-5	1011 0101
B6				12-11-0-6	1011 0110
B7				12-11-0-7	1011 0111
B8				12-11-0-8	1011 1000
B9				12-11-0-9	1011 1001
BA				12-11-0-8-2	1011 1010
BB				12-11-0-8-3	1011 1011
BC				12-11-0-8-4	1011 1100
BD				12-11-0-8-5	1011 1101
BE				12-11-0-8-6	1011 1110
BF				12-11-0-8-7	1011 1111
C0		?		12-0	1100 0000
C1		A	A	12-1	1100 0001
C2		B	B	12-2	1100 0010
C3		C	C	12-3	1100 0011
C4		D	D	12-4	1100 0100
C5		E	E	12-5	1100 0101
C6		F	F	12-6	1100 0110
C7		G	G	12-7	1100 0111
C8		H	H	12-8	1100 1000
C9		I	I	12-9	1100 1001
CA				12-0-9-8-2	1100 1010
CB				12-0-9-8-3	1100 1011
CC				12-0-9-8-4	1100 1100
CD				12-0-9-8-5	1100 1101
CE				12-0-9-8-6	1100 1110
CF				12-0-9-8-7	1100 1111
D0		!		11-0	1101 0000
D1	MVN	J	J	11-1	1101 0001
D2	MVC	K	K	11-2	1101 0010
D3	MVZ	L	L	11-3	1101 0011
D4	NC	M	M	11-4	1101 0100
D5	CLC	N	N	11-5	1101 0101
D6	OC	O	O	11-6	1101 0110
D7	XC	P	P	11-7	1101 0111
D8		Q	Q	11-8	1101 1000
D9		R	R	11-9	1101 1001
DA				12-11-9-8-2	1101 1010
DB				12-11-9-8-3	1101 1011
DC	TR			12-11-9-8-4	1101 1100
DD	TRT			12-11-9-8-5	1101 1101
DE	ED (4)			12-11-9-8-6	1101 1110
DF	EDMK (4)			12-11-9-8-7	1101 1111
E0		‡		0-8-2	1110 0000
E1				11-0-9-1	1110 0001
E2		S	S	0-2	1110 0010
E3		T	T	0-3	1110 0011
E4		U	U	0-4	1110 0100
E5		V	V	0-5	1110 0101
E6		W	W	0-6	1110 0110
E7		X	X	0-7	1110 0111
E8		Y	Y	0-8	1110 1000
E9		Z	Z	0-9	1110 1001
EA				11-0-9-8-2	1110 1010
EB				11-0-9-8-3	1110 1011
EC				11-0-9-8-4	1110 1100
ED				11-0-9-8-5	1110 1101
EE				11-0-9-8-6	1110 1110
EF				11-0-9-8-7	1110 1111
F0		0	0	0	1111 0000
F1	MVO	1	1	1	1111 0001
F2	PACK	2	2	2	1111 0010
F3	UNPK	3	3	3	1111 0011
F4		4	4	4	1111 0100
F5		5	5	5	1111 0101
F6		6	6	6	1111 0110
F7		7	7	7	1111 0111
F8	ZAP (4)	8	8	8	1111 1000
F9	CP (4)	9	9	9	1111 1001
FA	AP (4)			12-11-0-9-8-2	1111 1010
FB	SP (4)			12-11-0-9-8-3	1111 1011
FC	MP (4)			12-11-0-9-8-4	1111 1100
FD	DP (4)			12-11-0-9-8-5	1111 1101
FE				12-11-0-9-8-6	1111 1110
FF				12-11-0-9-8-7	1111 1111

Figure 39. IBM System/360 Eight-Bit Code

The 4-7 recording positions of nine-track tape parallel the function of the 8, 4, 2, 1 bit positions of seven-track tape. The 2 and 3 recording positions are the exact reverse of the A and B bit positions of seven-track. Positions 0 ond 1 are the two additional recording channels that group the characters into one of four classifications: uppercase alpha and numeric, lowercase alpha, special characters, and no assigned character. Note that the actual channels on nine-track tape do not run in 0-7 sequence.

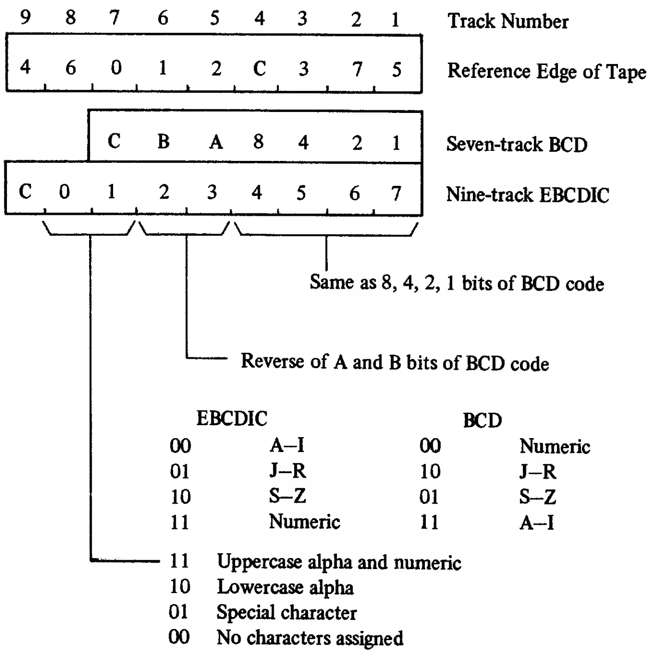


Figure 40. Comparison of seven-track and nine-track alphabetic code

Character Reading

Magnetic Ink Characters

Another method of representing data on paper media for machine processing is with magnetic ink characters — a language readable by both man and machine. Magnetic ink characters are printed on paper, as in Figure 41. The shape of the characters permits easy visual interpretation; the special magnetic ink allows reading or interpretation by machine.

The printing (inscribing) of magnetic ink characters on the paper documents is done by machine. The paper documents may be random size paper or cards ranging from 2¾ inches to 3¾ inches wide, from 6 inches to 8¾ inches long, and from .003 inch to .007 inch thick.

The IBM 1260 Electronic Inscriber, in addition to performing the normal proving functions related to banking procedures. inscribes documents. After in-

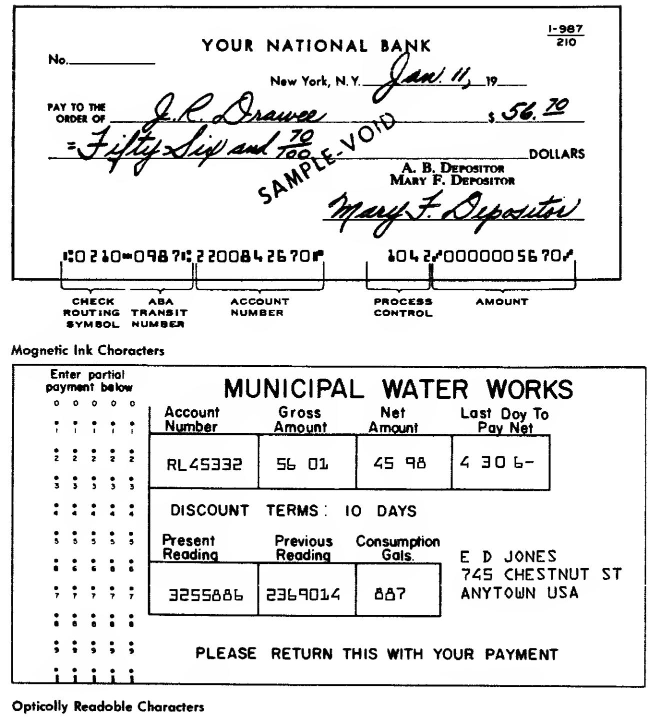


Figure 41. Magnetically and optically readable characters

scription, the IBM 1419 Magnetic Character Reader reads the inscribed information from the documents and converts it to a machine language. At this point, the information enters directly into an IBM data processing system. The 1419 can sort the documents as well.

Optically Read Characters

Another method of representing data on paper documents for input to a data processing system uses optically readable characters. Figure 41 shows some of the characters acceptable to the 1418/1428 type of optical reader. They include the 26 letters of the alphabet, digits 0 through 9, and special characters.

The 1231 Optical Mark Page Reader can also read ordinary No. 2 pencil marks or printed marks from 1403 or 1443 Printers on common (8½ x 11") sheets of paper.

The IBM 1285 reads printed paper tapes, such as those produced on cash registers and adding machines. Primary applications of optical reading are utility billing, insurance premium notices, charge sales invoices, etc.

The 1287 Optical Reader can read hand-printed or machine printed numeric digits and certain alphabetic characters from paper or card documents or journal tapes.

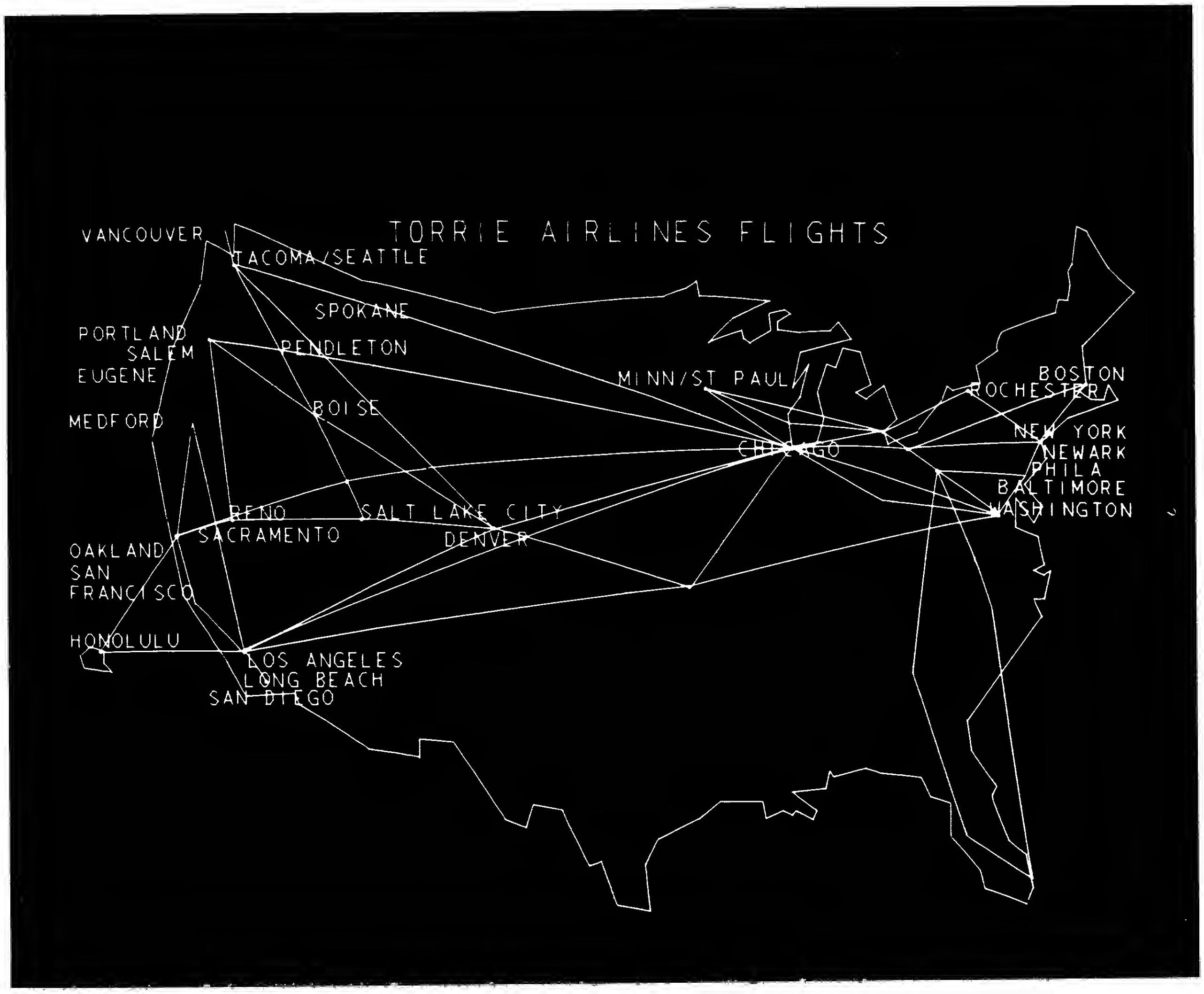


Figure 42. IBM 2250 screen contents. A display including absolute vector graphics, point plotting, and both sizes of alphabetic characters.

Visual Output

Visual display units in several sizes, capacities, speeds, and capabilities to handle complexities of information permit the user of a computing system to see, on a cathode ray tube, graphic reports that would take many times longer to produce by normal printing methods. The use of a visual display unit as a system operator console is a typical application. Another is the retrieval and presentation of a client's account record during a telephone inquiry. It is possible to update the record immediately (by using an entry keyboard) and return the corrected data to storage.

The display units present (on the cathode ray tube screen) tables, graphs, charts, and alphameric letters and figures. The IBM 2250 Display Unit has a display area containing over one million points that can be addressed by X and Y coordinates. It can display 52 lines of 74 characters each – all on a twelve-inch-square area. Figure 42 shows an example of a display on the 2250.

The 2840 Control Unit, connected to the 2250, accepts and stores data from the computer at up to 238,000 characters per second. As many as 60,000 characters, or lines, can be displayed per second. Horizontal and vertical lines may be drawn by specifying only the end points of the lines; in addition, a special feature enables lines to be drawn at any angle. Points may be displayed as fast as 16.8 microseconds (millionths of a second) per point.

The 2260 Display Unit can display twelve lines, of 80 characters, on its 4 x 9″ display area.

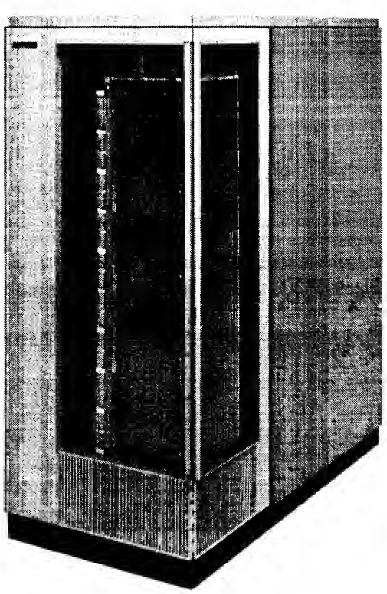
The 2848 Control Unit, connected to the 2260, accepts and stores data from the computer at rates of up to 2560 characters per second.

Storage Devices

Several types of IBM storage are presently available: core, magnetic drum, magnetic disk, and magnetic-strip data cells (Figure 43). Sometimes magnetic tape is thought of as storage rather than as an input/output medium.

IBM 2303 Drum Storage

IBM 2321 Data Cell Drive Model 1



IBM 2361 Core Storage

IBM 2311 Disk Storage Drive

Figure 43. IBM storage devices

Information can be placed into, held in, or removed from, computer storage as needed. The information can be:

1. Instructions to direct the central processing unit
2. Data (input, in-process, or output)
3. Reference data associated with processing (tables, code charts, constant factors, and so on).

Storage is classified as main or auxiliary, as in System/360 (Figure 44). Main storage includes all core storage.

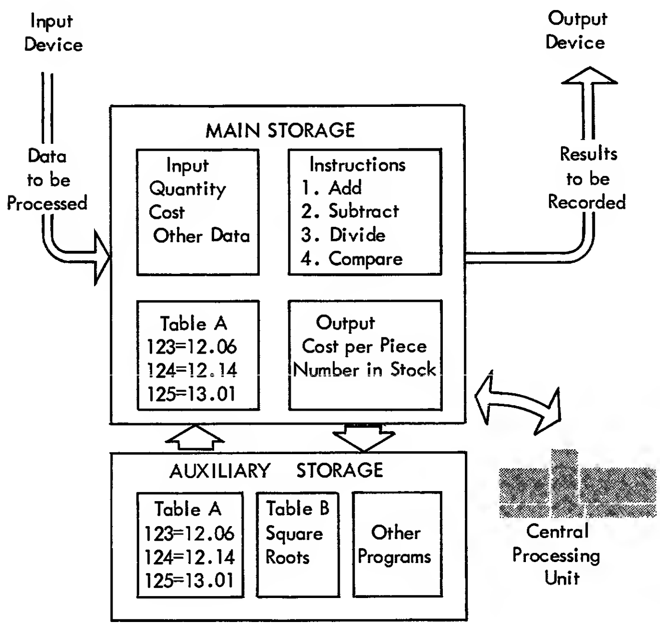


Figure 44. Schematic, main, and auxiliary storage

Auxiliary refers to all other storage and is of two types:

1. *Direct access.* Drum, disk, and data cell devices in which records can be accessed without having to read from the beginning of a file to find them.
2. *Sequential.* Tape units and Hypertape, where reels must be read from the beginning in order to read or write a desired record.

Main storage accepts data from an input unit, exchanges data with and supplies instructions to the central processing unit, and can furnish data to an output unit. All data to be processed by any system must pass through main storage. This unit must therefore have capacity to retain a usable amount of data and the necessary instructions for processing.

Applications can require additional storage. If so, the capacity of main storage is augmented by an auxiliary storage unit. All information to and from auxiliary storage must be routed through main storage.

Storage is arranged somewhat like a group of numbered mail boxes in a post office (Figure 45).

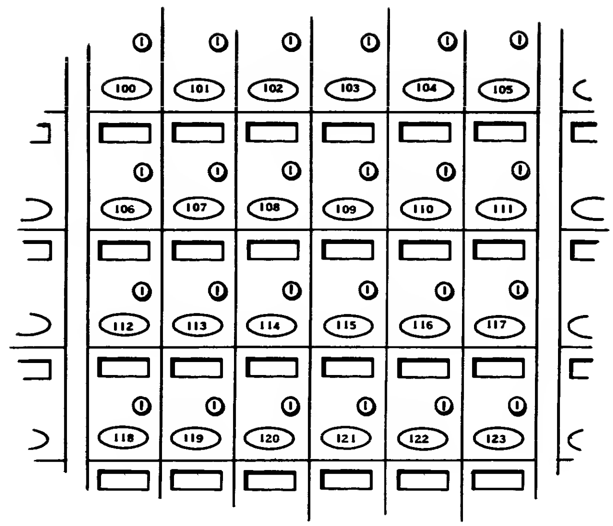


Figure 45. Post office mail boxes

Each box is identified and located by its number. In the same way, storage is divided into locations, each with an assigned address. Each location holds a specific unit of data. Depending on the system, the unit of data may be a character, a digit, an entire record, or a word. To insert or remove data at a location, the address must be known, either to the programmer or to a control program (explained later).

When information enters a location, it replaces the previous contents of that location. However, when information is taken from a location, the contents remain unaltered. Thus, once located in storage, the same data may be used many times. In effect, a duplicate of the information is made available for processing.

The computer requires some time to locate and transfer information to or from storage. This is called access time. Storage units are available whose access time is so brief that it is measured in billionths of a second. To appreciate such a minute interval of time, consider a spaceship of the future traveling at 100,000 miles per hour. In one-millionth of a second, the spaceship would travel about 1¾ inches. In a nanosecond (billionth of a second), it would travel about one-thousandth of 1¾ inches.

Because so many references must be made to storage in all data processing operations, the access speed has a direct bearing on the efficiency of the entire system.

For example, core storage is the most expensive storage device in terms of cost per storage location. However, core storage also provides the fastest access time; thus, it may be the most economical in terms of cost per machine calculation. Drum storage offers the advantages of lower direct cost to offset slower speed.

Most disk storage devices are slower than drum storage but offer the advantage of capacity in millions of digits. The largest single storage device is the data cell drive with a capacity of about 400 million eight-bit characters (or about 800 million four-bit numeric digits)!

Core Storage

A magnetic core is a tiny ring of ferromagnetic material, a few hundredths of an inch in diameter. Each core is pressed from a mixture of ferric oxide powder and other materials and then baked in an oven.

Aside from its compact size – a decided advantage in computer design – the important characteristic of the core is that it can be easily magnetized in a few millionths of a second. And, unless deliberately changed, it retains its magnetism indefinitely.

If cores are placed on a wire, like beads on a string, and a strong enough electrical current is sent through the wire, the cores become magnetized (Figure 46). The direction of current determines the polarity or magnetic state of the core (Figure 47). By reversing the direction of current, the magnetic state is changed (Figure 48). Consequently, the two states can be used to represent 0 or 1, plus or minus, yes or no, or on or off conditions. For machine purposes, this is the basis of the binary system of storing information. Because any specified location of storage must be instantly accessible, the cores are arranged so that any combination of ones (1s) and zeros (0s) representing a character can be written magnetically or read back when needed.

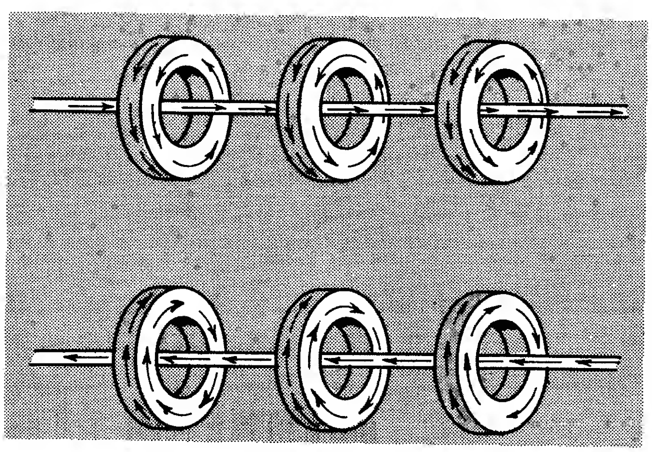


Figure 46. Polarity of magnetic cores

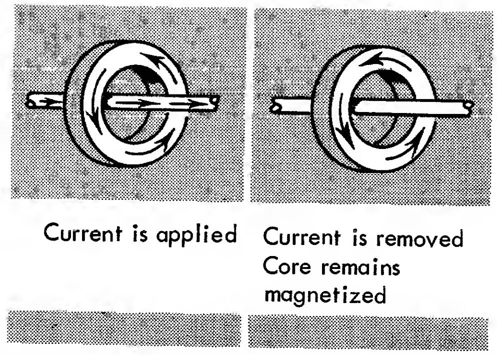


Figure 47. Magnetizing a core

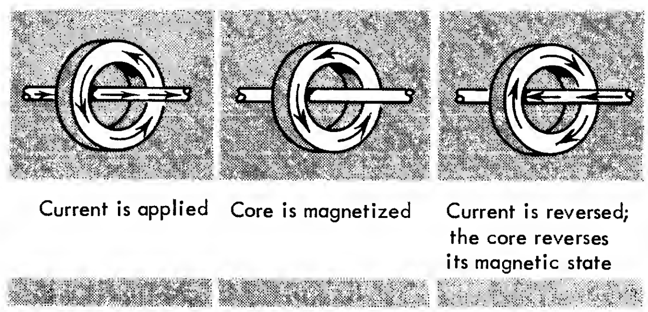


Figure 48. Reversing a core

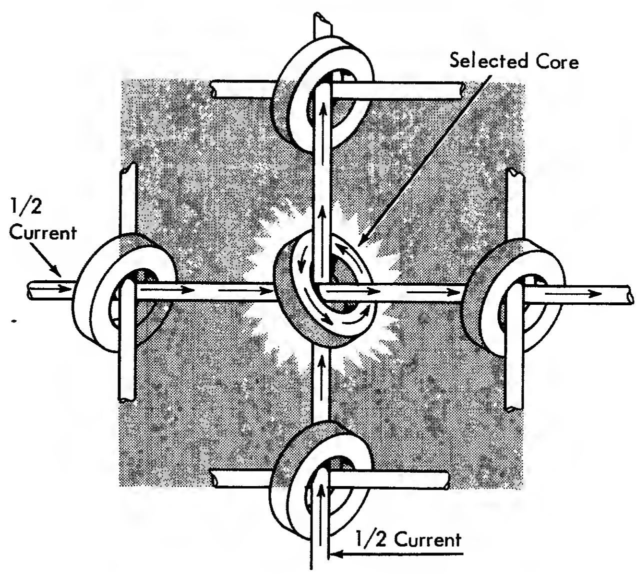


Figure 49. Selecting a core

To accomplish selection, two wires run through each core at right angles to each other (Figure 49). When half the current needed to magnetize a core is sent through each wire, only the core at the intersection of the wires is magnetized. No other core in the string is affected. Using this principle, a large number of cores can be strung on a screen of wires; yet, any single core in the screen can be selected for storage or reading without affecting any other.

Once information is placed in core storage, some means must be devised to make it accessible, that is, to recall it when needed. It has been shown that a definite magnetic polarity can be set up in a core by the flow of current through a wire. In the machine, the flow is not actually constant; it is sent through the wire as an electrical pulse, which is said to flip the core to a positive or a negative state, depending on the direction of current flow.

If the magnetic state of the core is reversed by the pulse, the abrupt change or flip induces current in a third wire running through the center of the core (Figure 50).

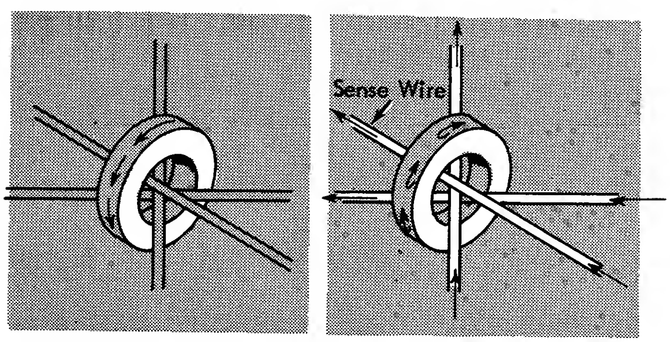


Figure 50. Core sense wire

The signal through this sense wire can be detected to determine whether the core contained a 1. Only one sense wire is needed for an entire core plane, because only one core at a time in any plane is tested for its magnetic state. The wire is therefore strung through all the cores of the plane (Figure 51).

Note, however, that when information has been read from storage, all cores storing that information are set to 0. Readout is destructive; that is, the process of reading a 1 resets the core to 0. Therefore, to retain data in storage, the computer must replace ones (1s) in those cores that had previously contained ones (1s). But cores that contained zeros (0s) must remain zeros (0s).

To reproduce (regenerate) ones (1s) as they should be, the computer tries to write back ones (1s) in all

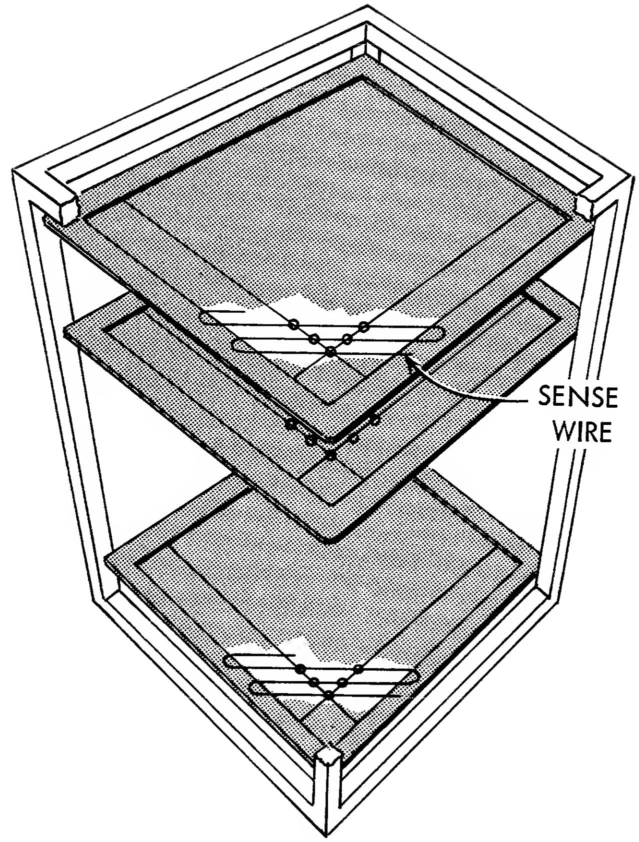


Figure 51. Sense wire in core plane

the locations previously read; at the same time, an inhibit pulse suppresses writing in cores that previously contained zeros (0s). The inhibit is sent through a fourth wire and, in effect, cancels out the writing pulse in one of the two wires used to magnetize the core. Like the sense wire, the inhibit wire (Figure 52) also runs through every core in a plane.

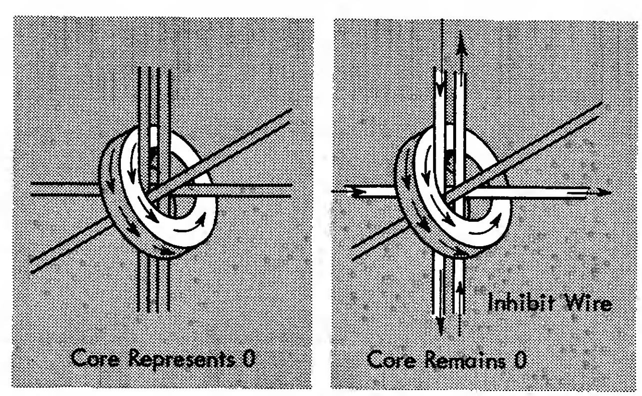


Figure 52. Core inhibit wire

It is beyond the scope of this manual to fully explain core storage. However, a basic knowledge of how core storage works is helpful in understanding the operation of all data processing systems using cores. In some core storage systems, the sense and inhibit functions are combined into one wire, which carries a charge in one direction for sensing and a charge in the opposite direction for inhibiting. Still more compact and advanced systems, such as that of the large-capacity IBM 2361 Core Storage shown in Figures 10 and 43, use only two wires and operate on a quite different principle – very time-saving and efficient, yet producing the same bit manipulation described here.

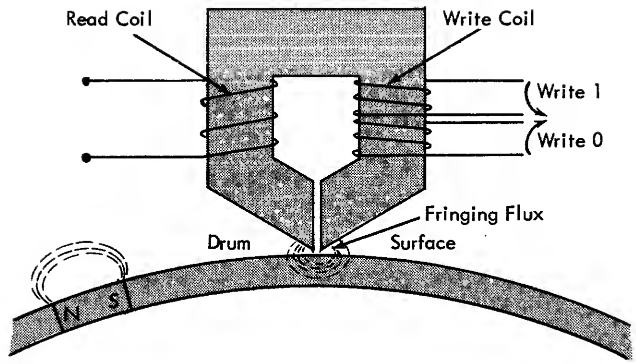


Figure 53. Drum recording

Magnetic Drum Storage

Two storage concepts are used in magnetic drum storage operations. The first is to use the drum as a high-capacity, intermediate-access, storage device. Principally, it is used for the storage of data that is referred to repeatedly throughout the computing operation (actuarial tables, logarithmic tables, etc.) or as a supplementary storage facility for main storage. The second and more recent concept is to provide program storage, program modification data, and a temporary storage for high-activity direct access operations involving limited amounts of data.

A magnetic drum is a cylinder that rotates at a constant speed and the outer surface of which is coated with a magnetic material. If an area of this material is placed in a magnetic field, the area becomes magnetized. After the magnetic field is removed, the magnetized spot remains on the surface of the drum indefinitely. Data recorded on the surface may be read repetitively. Each time new data is recorded, the old data is automatically erased. Information is recorded or retrieved by read/write heads that are suspended a very slight distance from the periphery of the drum. The read/write heads (Figure 53) contain coils of fine wire wound around tiny magnetic cores.

By sending pulses of current to the write coils in the read/write heads, the magnetic drum surface is magnetized. Conversely, by passing the magnetized spots recorded on the drum surface under the read coils, the drum-recorded data is used. Each drum has a specific number of storage locations, each of which is addressable by the computer. The capacity of each storage location depends upon the design of the drum and the data representation code used.

For an example of drum storage operation, let us consider the IBM 2303 Drum Storage, which consists of a vertically mounted drum and its associated electronic circuitry. The drum, coated with a magnetic recording material, rotates at about 3500 revolutions per minute. The surface of the drum is divided into addressable tracks, which extend around the periphery of the drum, and which are used for storing data as follows:

800 Standard Data Tracks
80 Alternate Data Tracks

The alternate tracks are provided to ensure that each recorded bit can be stored in a magnetically reliable medium. If a defect develops on a track, an alternate track is substituted, and the alternate is given the address of the disabled track.

Each data track has its own read/write head, which is used for both recording and retrieving data. The read/write heads are fixed in position on 20 vertical racks that surround the drum. Each rack contains 40 heads. If required, heads are readily moved (by the customer engineer) from disabled tracks to alternate tracks.

The read/write heads contain tiny, coil-wrapped, magnetic cores. During writing operations, these cores convert electrical signals, received from the computer, into magnetic flux to magnetize defined spots on the drum surface. During reading operations, the action is reversed; the magnetized spots on the drum surface generate a magnetic flux, which is converted to an electrical signal by the read/write head and transmitted to the computer.

Data Access Times

Two interrelated modes of data access are involved in drum storage operation, as compared to three in disk storage operation. One access mode is mechanical, the other electronic.

Magnetic Disk Storage

Disk storage, like drum storage, provides IBM data processing systems with the ability to record and retrieve stored data sequentially or randomly (directly). It permits immediate access to specific areas of information without the need to examine sequentially all recorded data. Magnetic tape operations do not have this ability; tape searching must start at the beginning of the tape reel and continue sequentially through all records until the desired information area is found.

For an example of the application of direct access operations, as compared to sequential operations, consider the search for a word in a large unabridged dictionary. If the contents of the dictionary were stored on magnetic tape, the complete dictionary could be machine-read in about two minutes. A wide range of individual words would require an average of one minute to be found and read by the magnetic tape sequential method of searching. Using the dictionary, a human being would average about 1/5 of a minute per word, simply because he would limit his search for each word to an appropriate portion of the whole dictionary. That is, he would immediately go to a specific letter rather than start at the beginning of the dictionary and check each entry. This concept of limiting a search to a small section of the whole would permit direct access storage to perform the dictionary word search in a few thousandths of a second.

The high-speed access to data storage locations provided by direct access data processing permits the user to maintain up-to-date files and to make frequent direct reference to the stored data.

The magnetic disk is a thin metal disk coated on both sides with magnetic recording material. Disks are mounted on a vertical shaft; they are slightly separated from one another to provide space for the movement of read/write assemblies. The shaft revolves, spinning the disk (Figure 54).

Data is stored as magnetized spots in concentric tracks on each surface of the disk. Some units have 500 tracks on each surface. The tracks are accessible for reading and writing by positioning the read/write heads between the spinning disks.

On the IBM 2302 Disk Storage, the read/write heads are mounted on an access mechanism with 24 arms, arranged like teeth on a comb, that move horizontally between the disks. Two read/write heads are mounted on each arm. One head services the bottom surface of the upper disk; the other head services the top surface of the lower disk. Thus, it is possible to read or to write on either side of the disk.

The magnetic disk data surface can be used repetitively. Each time as new information is recorded and stored on a track, the old information is erased. The recorded data may be read as often as desired; data remains recorded until written over.

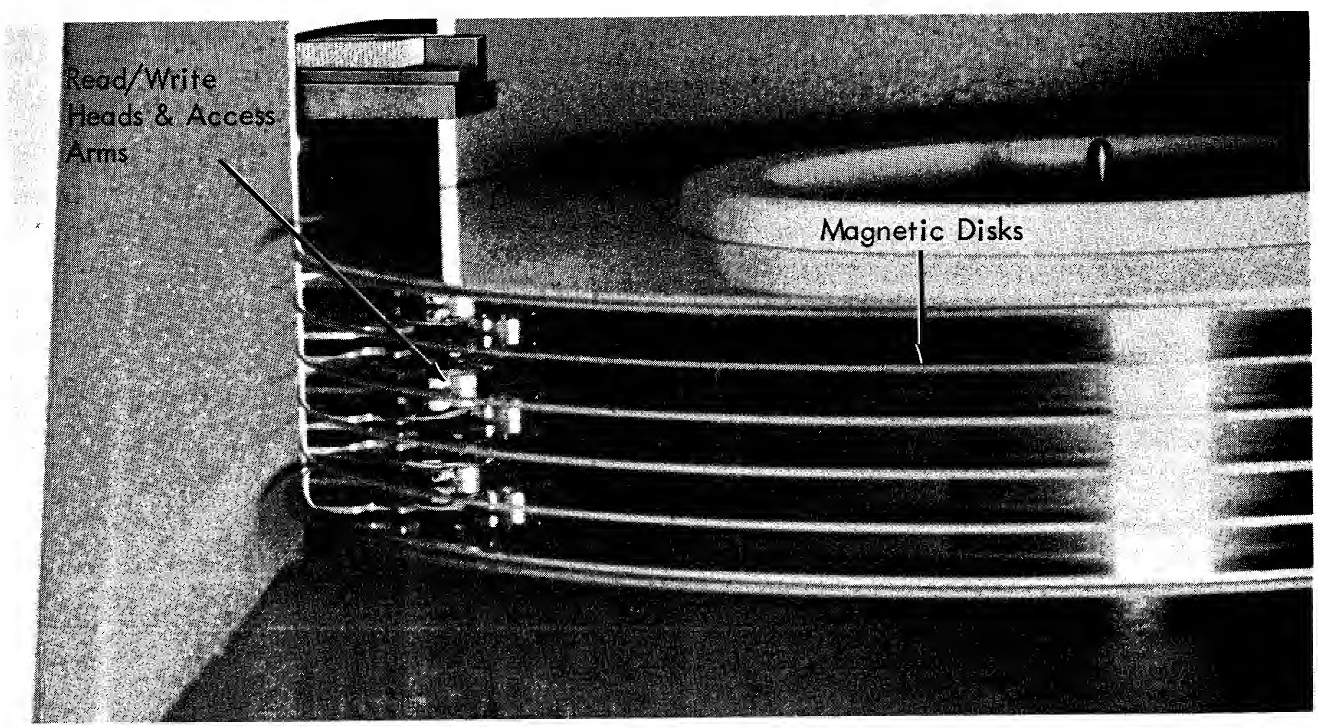


Figure 54. IBM 2311 Disk Storage Drive showing mounted disk pack and access arms

The IBM 2302 Disk Storage contains one or two modules of disk assemblies. Each module consists of 25 magnetically coated disks, two feet in diameter, and an access mechanism. Each disk surface has 492 tracks. The disks are mounted ½ inch apart on the rotating vertical shaft.

The IBM 2311 Disk Storage (Figure 54) is similar in operating principle to the 2302, except that the 2311 uses interchangeable disk packs. Six disks are mounted as a disk pack which can be readily removed from the 2311 Disk Drive and stored in a library of disk packs in much the same manner as reels of magnetic tape may be stored. The packs are 14 inches in diameter and weigh less than ten pounds. Each of the ten recording surfaces contains 200 data recording tracks. The disks turn at 2400 revolutions per minute. Up to 7.25 million characters of information can be stored on each disk pack.

The IBM 2314 Direct Access Storage Facility consists of nine drives and a control unit. Any eight of the drives can be online at a time. The ninth drive is available for backup if one of the other drives requires servicing or maintenance. The device uses removeable disk packs similar to those on the 2311. The packs are larger, however, each consisting of eleven disks, with 20 of the surfaces used for recording. Each surface has 200 data recording tracks. Up to 29.18 million bytes of information can be stored on each disk pack.

Figure 55. IBM 2321 Data Cell Drive

Data Cell Storage

A data cell shown in Figure 55 stores several hundred strips of magnetic film approximately two inches wide and twelve inches long. Ten of these strips grouped are called a sub-cell. Twenty sub-cells in turn make up one data cell. Individual strips can be retrieved by the device and information stored on the 200 tracks available on each strip.

The cell drive can accommodate up to ten data cells and positions the selected cell under the retrieval mechanism. Cells may be removed and replaced with others containing different files. Each of the drives shown in Figure 56 has a capacity for storing up to 40 million characters.

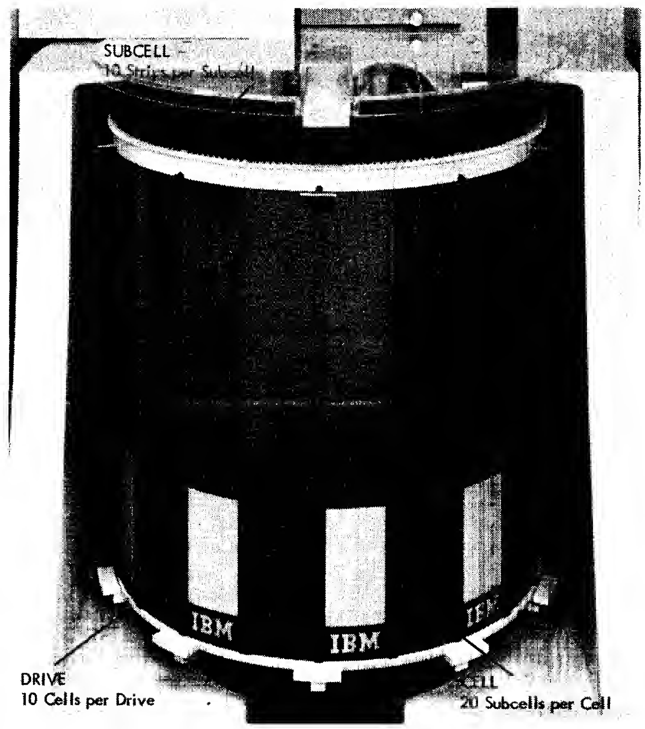


Figure 56. IBM 2321 Data Cell Drive, Cell, and Sub-cell

The data cell drive economically extends online direct access storage capabilities to a volume of data beyond that of other storage devices. Each drive offers 400 million characters of data.

Storage and Data Processing Methods

IBM data processing systems use two methods of data handling – sequential or batch processing and inline or direct access processing (see Figure 57). The application requirements determine which method is needed.

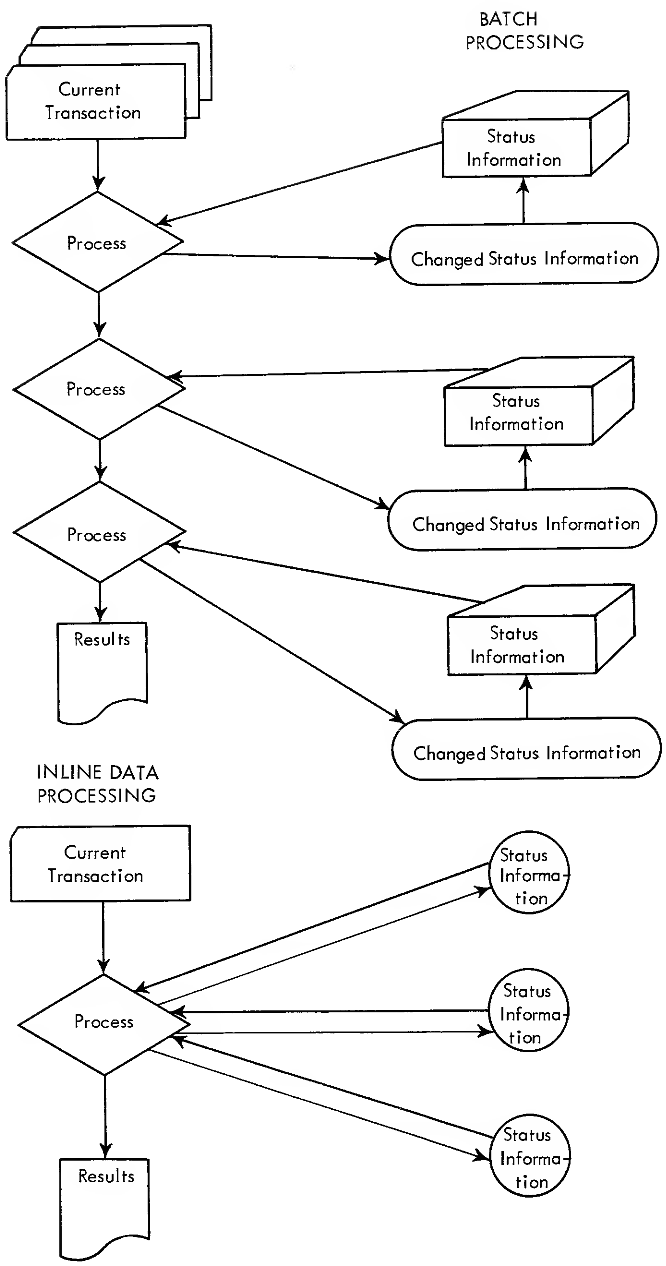


Figure 57. Batch and direct access processing

In either case, all data pertaining to a single application is maintained in files (often called data sets).

In sequential processing, these files are stored outside the computer — usually on magnetic tape — and they are arranged in a predetermined sequence. The data may concern inventory, accounts receivable, accounts payable, payroll, and the like. Each file (data set) is made up of records, each containing information required to describe completely a single item. The sequence may be by item number, name, account number, or man number, but all files pertaining to a single application must be in the same sequence.

In many cases, processing involves not only performing calculation on some parts of each record to arrive at balances, amounts, or earnings, but also involves adding, changing, or deleting records as new transactions occur. However, before transactions can be applied against the main or master file, they must also be arranged in the same sequence as the master file. For this reason, they are accumulated in convenient groups or batches.

The two files (data sets), master and transaction, now become input to the data processing system. One record or a small group of records (also called a block) is read into storage at a time. These are processed, and the result is written as output. When magnetic tape files are used, the output records with the updated results of current processing must be recorded on a separate tape, producing a new master that will be used as input the next time the job is to be done. The next group of records is read in, and the process is repeated. The series of repetitive operations continues under the direction of program instructions, record by record, until the input files are exhausted. The results form a revised master file, updated according to the current transactions. The new master file is in the same sequence as the original files.

Other output may also be produced as a by-product of the processing. This output may be records of delinquent accounts, bank orders, earnings statements, payroll checks, and so on. In every case, however, the sequence of all output remains the same as the sequence of the incoming data.

With sequential processing, the information in storage is transient. Consequently, the storage unit needs only enough capacity for program instructions, plus the largest element of data to be processed.

When direct access processing is used, transactions affecting the contents of the file (data set) are fed to the computer directly, as they occur. In this case, the computer locates the corresponding record or data in storage and adjusts this master record accordingly. Accounts or balances are constantly maintained and are available as output when needed. Transactions are not batched, and they need not be sorted before processing.

Central Processing Unit (CPU)

The central processing unit controls and supervises the entire computer system and performs the actual arithmetic and logical operations on data. From a functional viewpoint, the CPU consists of two sections – control and arithmetic/logical (Figure 58).

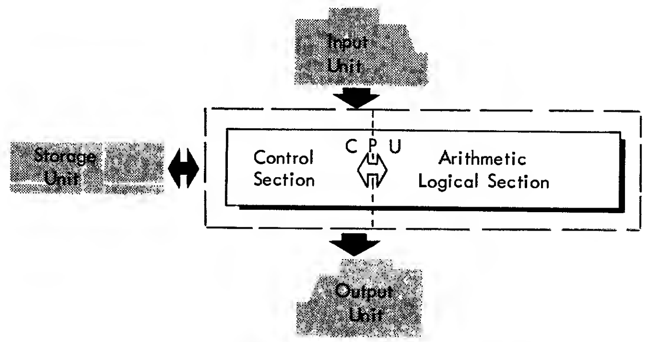


Figure 58. Central processing unit in the data processing system

The control section directs and coordinates all operations called for by instructions. This involves control of input/output devices, entry or removal of information from storage, and routing of information between storage and the arithmetic/logical section. Through the action of the control section, automatic, integrated operation of the entire computer system is achieved.

In many ways, the control section can be compared to a telephone exchange. All possible data transfer paths already exist, just as there are connecting lines between all telephones serviced by a central exchange (Figure 59).

The telephone exchange has a means of controlling instruments that carry sound pulses from one phone to another, ring the phones, connect and disconnect circuits, and so on. The path of conversation between one telephone and another is set up by appropriate controls in the exchange itself. In the computer, execution of an instruction involves opening and closing many paths or gates for a given operation. Some functions of the control section are to start or stop an input/output unit, to turn a signal device on or off, to rewind a tape reel, or to direct a process of calculation. In some System/360 models, a part of this section consists of a control device called "read-only storage" that contains circuits for performing operations designated by the operation codes. It also houses the emulator circuits that the user may select to make his System/360 perform programming instructions written for other computers.

The arithmetic/logical section contains the circuitry to perform arithmetic and logical operations. The former portion calculates, shifts numbers, sets the algebraic sign of results, rounds, compares, and so on. The latter portion carries out the decision-making operations to change the sequence of instruction execution.

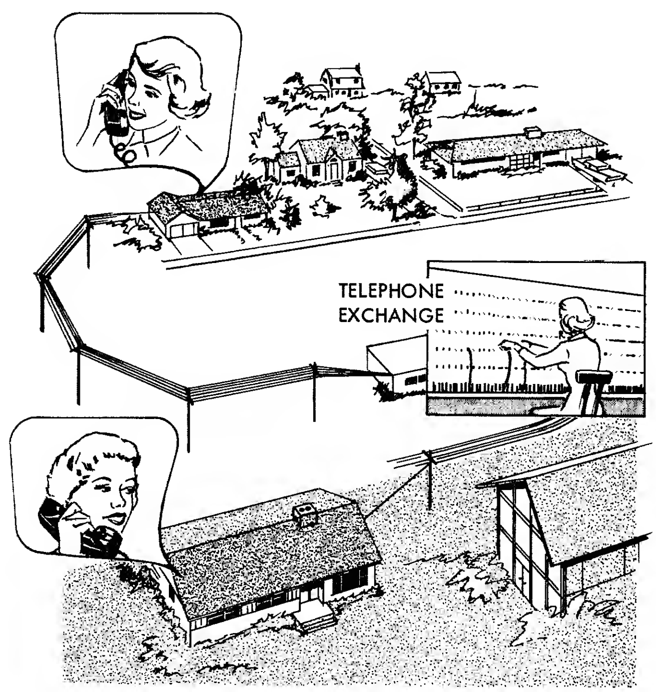


Figure 59. Telephone exchange system

Functional Units

Register

A register is a device capable of receiving information, holding it, and transferring it as directed by control circuits. The electronic components used may be magnetic cores or transistors.

Registers are named according to their function: an accumulator accumulates results; a multiplier-quotient holds either multiplier or quotient; a storage register contains information taken from or being sent to storage; an address register holds the address of a storage location or device; and an instruction register contains the instruction being executed (Figure 60). System/360 has general purpose registers, which are used for several functions, including storage addresses, index addresses, and data that is to be processed logically or arithmetically.

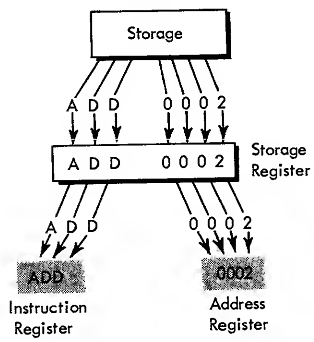


Figure 60. Register nomenclature and function

Registers differ in size, capacity, and use. In some cases, extra positions detect possible overflow conditions during an arithmetic operation. For example, if two eleven-digit numbers are added, it is possible that the result is a twelve-digit answer (Figure 61).

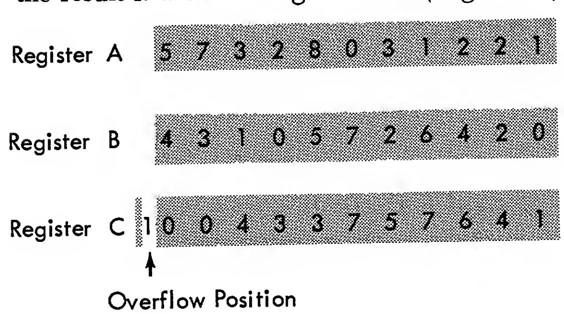


Figure 61. Overflow condition resulting from addition

In this figure, register A holds one factor, and register B holds the other factor. The two factors are combined, and the result is placed in register C, where an overflow condition is indicated by the presence of data in the overflow position. The contents of other registers can be shifted right or left within the register and, in some cases, even between registers. Figure 62 shows shifting of register contents three positions to the right. Positions vacated are filled with zeros, and numbers shifted beyond register capacity are lost.

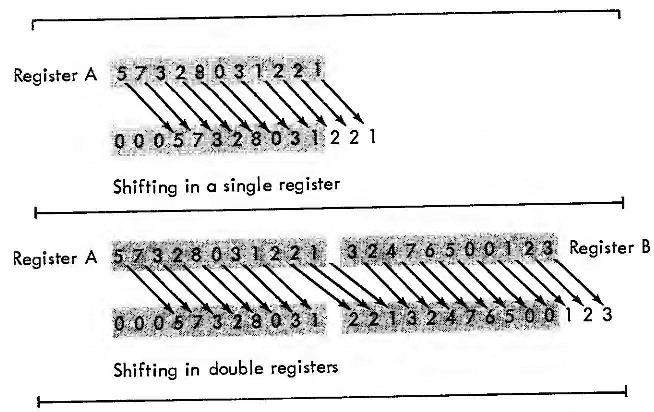


Figure 62. Types of computer register shifting

In other instances, a register holds data while associated circuits analyze the data. For example, an instruction can be placed in a register, and associated circuits can determine the operation to be performed and locate the data to be used. Data within specific registers may also be checked for validity.

The more important registers of a system, particularly those involved in normal data flow and storage addressing, have small lights associated with them. These lights are located on the machine console (Figure 63) for visual indication of register contents and various program conditions.

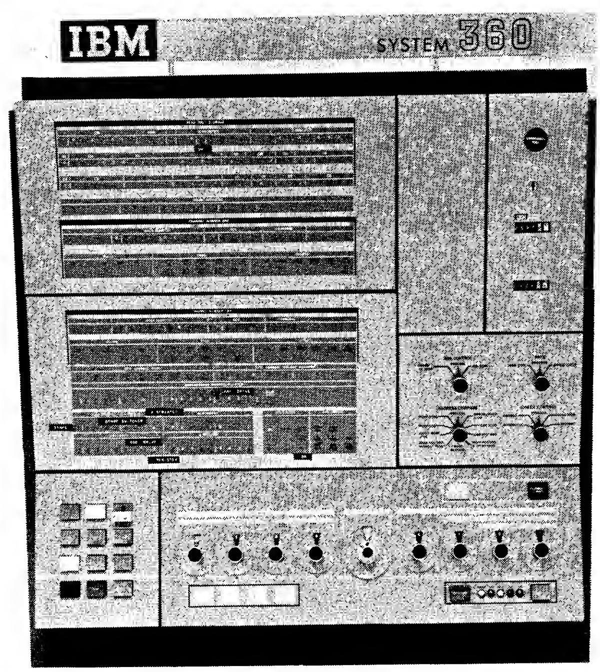


Figure 63. Typical operator console

Counter

The counter is closely related to a register, and may perform some of the same functions. Its contents can be increased or decreased. The action of a counter is related to its design and use within the computer system. Like the register, it may also have visual indicators on the machine console.

Adder

The adder receives data from two or more sources, performs addition, and sends the result to a receiving register or accumulator. Figure 64 shows two positions of an adder circuit, with input from registers A and B. The sum is developed in the adder. A carry from any position is sent to the next-higher-order position. The final sum goes to the corresponding positions of the receiving register.

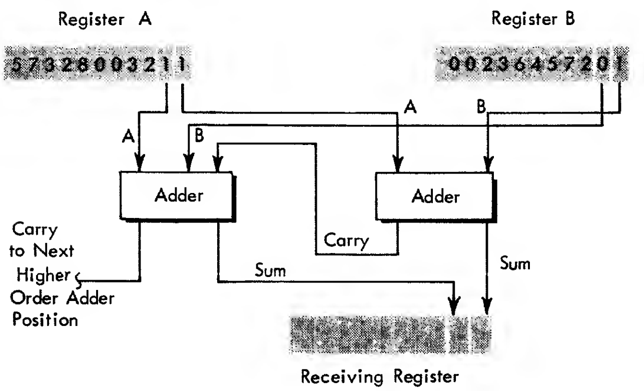


Figure 64. Adders in a computer system

Machine Cycles

All computer operations take place in fixed intervals of time. These intervals are measured by regular pulses emitted from an electronic clock at frequencies as high as five million per second. A fixed number of pulses determines the time of each basic machine cycle.

In computer usage, time references are stated in such terms as milliseconds, microseconds, and nanoseconds. These terms may convey no meaning unless it is realized just how short an interval a millisecond is. For example, the blink of an eye takes about one-tenth of a second or – 100 milliseconds!

The following table establishes some additional terms and abbreviations:

.1	= 1/10 second
	= 100 milliseconds
.001	= 1/1,000 second
	= 1 millisecond (ms)
.000001	= 1/1,000,000 second
	= 1 microsecond (μsec)
.000000001	= 1/1,000,000,000 second
	= 1 nanosecond (ns)

Within a machine cycle, the computer can perform a specific machine operation. The number of operations required to execute a single instruction depends on the instruction. Various machine operations are thus combined to execute each instruction.

Instructions usually consist of at least two parts, an operation and an operand. The operation tells the machine which function to perform: read, write, add, subtract, and so on. The operand can be the address of data or an instruction in main storage, the address of data or programs in secondary storage, or the address of an input/output unit. It can also specify a control function, such as shifting a quantity in a register, or backspacing and rewinding a reel of tape.

To receive, interpret, and execute instructions, the central processing unit must operate in a prescribed sequence, which is determined by the specific instruction and is carried out during a fixed interval of timed pulses.

Instruction Cycle

The first machine cycle required to execute an instruction is called an instruction cycle. The time for this cycle is instruction or I-time. During I-time:

1. The instruction is taken from a main storage location and brought to the central processing unit.
2. The operation part is decoded in an instruction register. This tells the machine what operation is to be performed.
3. The operand is placed in an address register. This tells the machine what factors are to be used in the operation.
4. The location of the next instruction to be executed is determined.

At the beginning of a program, an instruction counter is set to the address of the first program instruction. This instruction is brought from storage, and, while it is being executed, the instruction counter automatically advances (steps) to the location corresponding to the space occupied by the next stored instruction. If each instruction occupies one storage position, the counter steps one; if an instruction occupies five positions, the counter steps five. By the time one instruction is executed, the counter has located the next instruction in program sequence. The stepping action of the counter is automatic. In other words, when the computer is directed to a series of instructions, it executes them one after another until instructed to do otherwise.

Assume that an instruction is given to add the contents of storage location 2 to the contents of the accumulator register. Figure 65 shows the main registers involved and the information flow lines.

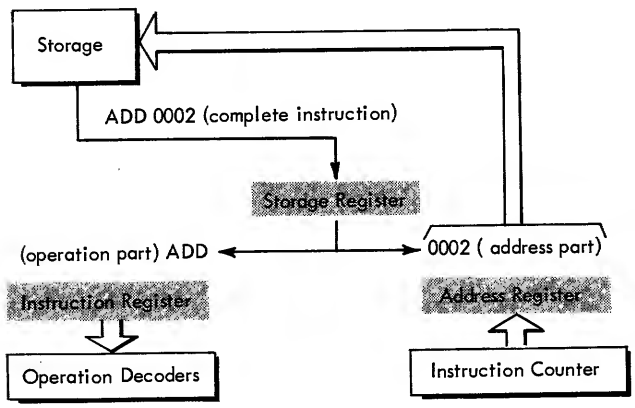


Figure 65. Computer I-cycle flow lines

I-time begins when the instruction counter transfers the location of the instruction to the address register. This instruction is selected from storage and placed in a storage register. From the storage register, the operation part is routed to the instruction register and the operand to the address register. Operation decoders then condition proper circuit paths to perform the instruction.

Execution of instructions does not necessarily have to proceed sequentially. Certain instructions alter the process of sequential execution unconditionally. In this case, an instruction brought from storage indicates that the next sequential instruction is not to be executed but that one located in another position is next. The normal stepping of the instruction counter can also be reset back to the beginning of the program so that the entire program can be repeated for another incoming group of data.

This branching (transfer) to alternative instructions may also be conditional. The computer can be directed to examine some indicating device and then branch if the indicator is on or off. Such an instruction could say, in effect, "Look at the sign of the quantity in the accumulator; if the sign is minus, take the next instruction from location 5000; if the sign is plus, proceed to the next instruction in sequence." The instruction counter is set according to one of the two possible storage locations (5000 or the location of the next instruction in sequence). The logical path followed by the computer (that is, the precise sequence of instructions executed) may be controlled either by unconditional branching or by a series of conditional tests applied at various points. However, the arrangement of instructions in storage is not normally altered.

Execution Cycle

I-time is usually followed by one or more machine cycles that occur during execution or E-time. The number of execution cycles required depends on the instruction to be executed. Figure 66 shows the data flow following I-time illustrated by Figure 65.

The E-cycle starts by removing from storage the information located at the address (0002) indicated by the address register. This information is placed in the storage register. In this case, one of the factors to be added is placed in the adders together with the number from the accumulator. The contents of the storage register and the accumulator are combined in the adders, and the sum is returned to the accumulator.

The address register may contain information other than the storage location of data. It can indicate the address of an input/output device or a control function to be performed. The operation part of the instruction tells the computer how to interpret this information.

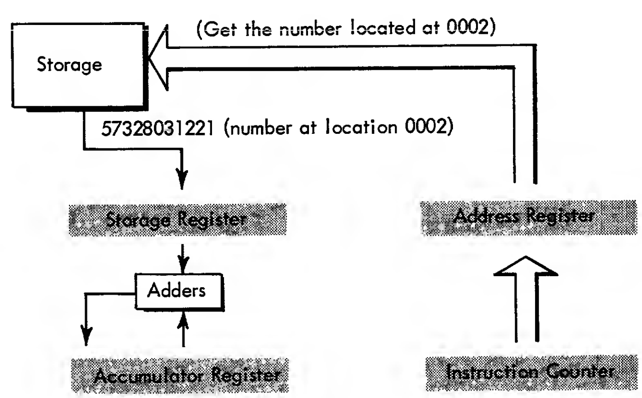


Figure 66. Computer E-cycle following an I-cycle

Serial and Parallel Operation

Computers and portions of computers are classified as either serial or parallel, depending on the method used to perform arithmetic. Essentially, all arithmetic is performed by addition.

In a serial computer, numbers to be added are considered one position at a time (hundred, etc.), in the same way that addition is done with paper and pencil. Whenever a carry is developed, it is retained temporarily and then added to the sum of the next-higher-order position.

The time required for serial operation depends on the number of digits in the factors to be added. Serial addition is shown in Figure 67.

	1st Step	2nd Step	3rd Step	4th Step
Addend	1234	1234	1234	1234
Augend	2459	2459	2459	2459
Carry	1	1		
Sum	3	93	693	3693

Figure 67. Serial addition

In a parallel computer, addition is performed on complete data words. The words are combined in one operation, including carries. Any two data words, regardless of the magnitude of the numbers contained in the words, can be added in the same time. Figure 68 shows parallel addition.

	00564213
Numbers being added	00000824
Carry	1
Final Result	00565037

Figure 68. Parallel addition

Fixed-Length and Variable-Length Words

Data can be addressed and processed by a computer system using either fixed-length or variable-length words.

In operations using fixed-length words, information is handled and addressed in units or words containing a predetermined number of positions. The size of a word is designed into the system, and it normally corresponds to the smallest unit of information that can be addressed for processing in the central processing unit. Records, fields, characters, or factors are all manipulated in parallel as words; registers, counters, accumulators, and storage are designed to accommodate a standard word.

In operations using variable-length words, data-handling circuitry is designed to process information serially as single characters. Records, fields, or factors may be of any practical length within the capacity of the storage unit. Information is available by character instead of by word.

Operation within a given data processing system may be entirely fixed-length, entirely variable, or a combination.

Floating-Point Operation

Mathematicians and scientists use logarithms to simplify mathematical manipulations of very large numbers and very small fractions. Similarly, scientific computers use floating-point operation. With the advent of time-shared computers for diverse applications, floating-point operation in multipurpose computers is becoming commonplace. We will first explain the principles of floating-point using the 7090/7094 data processing systems. The basic principles apply (with adjustments for different word or field makeup and internal numbering system logic) to the business computers and, finally, to the multipurpose System/360.

All central processing units that handle floating-point arithmetic do it by converting numbers (integers, fractions, or improper fractions) into the exponential form on the right-hand side of the equation. For example:

$N = b^e \times f$ where:

N = number

b = numbering system base (such as 2 for binary system, 10 for decimal system)

e = exponent (power to which the numbering system base is raised to make the expression equal to N)

f = fraction (similar to the mantissa in a logarithm)

The base used depends on the internal numbering system. For 7090/7094, it is 2. For System/360, it is 16.

Because the base is fixed by the design of the computer, it is eliminated from the formula, which then reads:

N = exponent x fraction

The range of exponents that can be used depends on the space allowed and on whether a position in the computer is allocated to contain a sign (+ or −) for the exponent. The space allotment is:

Eight bits without sign in the 7090/7094, permitting a binary range of a 2^{127} (decimal exponent). These values are about equal to 10^{38} through 10^{-38}.

Seven bits without sign in System/360, with a hexadecimal range of 16^{63} through 16^{-64}. Translated into the decimal system, this approximates a range of from 10^{76} down through 10^{-77}.

When no sign position is allowed for an exponent, a value in the center of the total range of possible exponents is chosen to represent the exponent 0. Positive exponents are added to this "middle" value; negative exponents are subtracted from it. This "scaled exponent" is called a characteristic.

The range of fractions in computers using fixed-length words is determined by the space remaining in the word (or words), allocated to express the floating-point number, after the exponent and sign have been provided for. In System/360, the fraction field may have up to 6 or up to 14 hexadecimal digits (24 or 56 bits), depending upon whether one or two 32-bit words are to be used in carrying out the precision of the fraction. The central processing units of other computers using fixed-length words usually have just two alternatives, called single- or double-precision,

depending on whether one word or two words are designated to store the number. In the 7090/7094, a single-precision fraction is up to 27 bits long. In the 7070/7074, it is up to eight digits long.

The length of floating-point fractions in the computers that do not have fixed-length words are:

IBM 1410 – 2 through 97 digits

IBM 1710 – 2 through 18 digits

Floating-point is so called because, using the exponent x fraction method of expressing a number, we can vary the exponent and then "float" the decimal point (in decimal computers), the binary point (in binary computers), or the hexadecimal point (in hexadecimal computers) correspondingly – and still not change the value of the number (Figure 69).

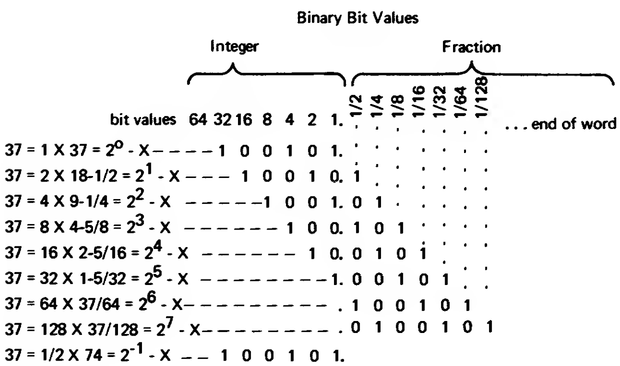


Figure 69. Different floating-point expressions of the same value

When the point falls just to the left of the high-order 1 bit (in binary) or the high-order significant digit (in decimal or hexadecimal) of the fraction, the floating-point expression is called normalized. When numbers are initially entered into the central processing unit of a computer as floating-point numbers, they take this normalized form. As arithmetic operations are performed on them, the points may float to the right or to the left as the computer seeks to get all the numbers involved in the operation into the same power of 2 (this is similar to changing fractions with different denominators into fractions with a common denominator in order to add them). The numbers then become decidedly unnormalized. Eventually, however, the computer normalizes the final result.

Binary Floating-Point Notation

Before proceeding, the subject of exponents and characteristics in binary computers must be clear.

The characteristics are not signed. Therefore, as stated before, the zero (0) exponent lies in the middle of the range of values possible in the characteristic field (eight bits). Since characteristics and fractions are usually expressed in octal, for easy comprehension and translation into binary, Figure 70 gives the relative decimal-octal values of the largest positive, zero, and largest negative exponents, and corresponding octal and binary characteristics.

	Decimal Exponent	Decimal Characteristic	Binary Characteristic
$10^{38} \approx 2^{127}$	127	245	11 111 111
$10^{0} \approx 2^{0}$	0	128	10 000 000
$10^{-38} \approx 2^{-128}$	-128	0	00 000 000

Figure 70. Range of characteristics in an IBM 1130 Computing System

In a binary system normalized floating-point number, the fraction always contains a high-order binary 1. Therefore, the fraction is always equal to, or more than, ½, but less than one. Figure 71 shows, with simple decimal system numbers, how to translate into normalized binary floating-point.

Decimal Representation			Characteristic Bit Positions	Fraction Bit Positions
Dec. No.	Exponent	Fraction		
$1 = 2^1 \times 1/2$	1	1/2	10 000 001	100 000 000
$2 = 2^2 \times 2$	2	1/2	10 000 010	100 000 000
$3 = 2^2 \times 3/4$	2	1/2+1/4	10 000 010	110 000 000
$4 = 2^3 \times 1/2$	3	1/2	10 000 011	100 000 000
$5 = 2^3 \times 5/8$	3	1/2+0/4+1/8	10 000 011	101 000 000
$10 = 2^4 \times 5/8$	4	1/2+0/4+1/8	10 000 100	101 000 000
$.5 = 2^0 \times 1/2$	0	1/2	10 000 000	100 000 000
$.005 = 2^{-2} \times 1/2$	-2	1/2	01 111 110	100 000 000

Figure 71. Examples of normalized binary floating-point numbers

An easy way to convert from decimal system integers into this combination of exponents of 2 and fractions is as follows:

1. Convert the decimal number to octal (see octal system). For example, the octal for 37 is 45.
2. Convert octal to binary (the binary for octal 45 is 100 101).
3. Place the binary point (it can no longer be called a decimal point, since we are now in the binary system) to the right of the binary number (hence, 100 101.).
4. Shift the binary point to the left of the leftmost 1 in the binary number (.100101).
5. The binary point is now where the vertical line appears in Figure 71. Put the binary number just to the right of that line (100101). It is now a fraction between ½ and 1.
6. Count the number of binary digits passed over when the binary point was shifted to the left (six digits). That number is the correct exponent of 2.

7. Add that number to the midpoint of the characteristic range, and enter it to the left of the vertical line. Example:

Characteristic	Fraction,	which means:
10000110	100101	

$2^6 \times 37/64$ or $64 \times 37/64 = 37$

Converting from decimal system fractions into exponents of 2 and fractions ($\geq$ ½ and < 1) is a similar procedure, except that we move the binary point to the right instead of to the left and increase the negative exponent of 2 in the process.

1. Convert the decimal fraction to octal. For example, decimal .145 is octal .1142+.
2. Convert octal to binary (the binary for octal 1142 is 001 001 100 010).
3. Place the binary point to the left of the binary number (.001 001 100 010).
4. Shift the binary point to the right so that it rests just to the left of the leftmost 1 in the binary fraction (00.100 110 001).
5. Count the number of binary digits passed over in shifting the binary point (two digits). That number is the negative exponent.
6. Subtract that number (in octal) from the midpoint of the characteristic range, and enter it to the left of the vertical line. (The octal value is 200 − 2 + 176.) Example:

Characteristic	Fraction
01 111 110	100 110 001

This $= 2^{-2} \times 1/2 + 1/16 + 1/32 + 1/512$
$= 1/4 \times [(256 + 32 + 16 + 1)/512]$
$= 1/4 \times 305/512$
$= 305/2048$
$= .1489+$ (or .149)

The computer manipulates these floating-point words for addition, as in the following example:

Add 12 + 97

Decimal Arithmetic	Octal Arithmetic
12	14
+97	+141
109_{10}	155_8

Floating-Point Binary
00.1 100 or $2^4 \times 1100$
+00.1 100 001 or $2^7 \times 1100001$

The floating-point words in the computer in normalized form are:

Characteristic (Octal)	Characteristic (Binary)	Fraction (Binary)
204	10 000 100	1100
207	10 000 111	1100001

The computer adjusts these two floating-point words to have a common characteristic and adds them.

Characteristic (Octal)	Characteristic (Binary)	Fraction (Binary)	
207	10 000 111	000 110 0	Add 3 to the characteristic, and add three high-order zeros to the fraction.
207	10 000 111	+110 000 1	Remains the same.
207	10 000 111	110 110 1	Translated ultimately into decimal, this is:

$2^7 \times (1/2 + 1/4 + 1/16 + 1/32 + 1/128) = 109$
$128 \times [(64 + 32 + 8 + 4 + 1)/128] = 109$

Since the simple addition example serves as an introduction to floating-point implementation within the computer, we will not undertake to show the more involved manipulation of fraction and characteristic that takes place in subtraction (complement addition), multiplication, and division.

Hexadecimal Floating-Point Notation

System/360, like the binary system computers, uses a point midway through the range of characteristics to express the exponent 0. The exponent is the power of 16 with which the fraction (also expressed in hexadecimal) must be multiplied to produce the desired value.

Converting a decimal number (assume the number 149.25) into a single floating-point word, as used in System/360, is done as follows:

1. Separate the number into the integer and the fraction;
   149.25 = 149 plus 0.25
2. Convert the decimal integer to hexadecimal;
   (see Figure 31) $149_{10} = 95_{16}$
3. Convert the decimal fraction to hexadecimal;
   (see Figure 32) $0.25_{10} = 0.4_{16}$
4. Combine the two and express in normalized form (as a fraction times an exponent to the base 16);
   $95.4_{16} = (0.954 \times 16^2)_{16}$
5. Since 64 is the midpoint of the characteristic range, add the exponent (2) to 64 to get the characteristic;
   64 + 2 = 66 = 1000010
6. Convert the fraction to binary, and group it hexadecimally;
   $.954_{16} = .1001\ 0101\ 0100$
7. The floating-point word for decimal 149.25 (hexadecimal 95.4) appears as:
   Sign* Characteristic
   0 1 100 0010
   Fraction
   1001 0101 0100 0000 0000 0000

*Zero (0) is used, since the value of 149.25 is positive.

An input/output unit is a device for putting in or getting out data from storage (Figure 72).

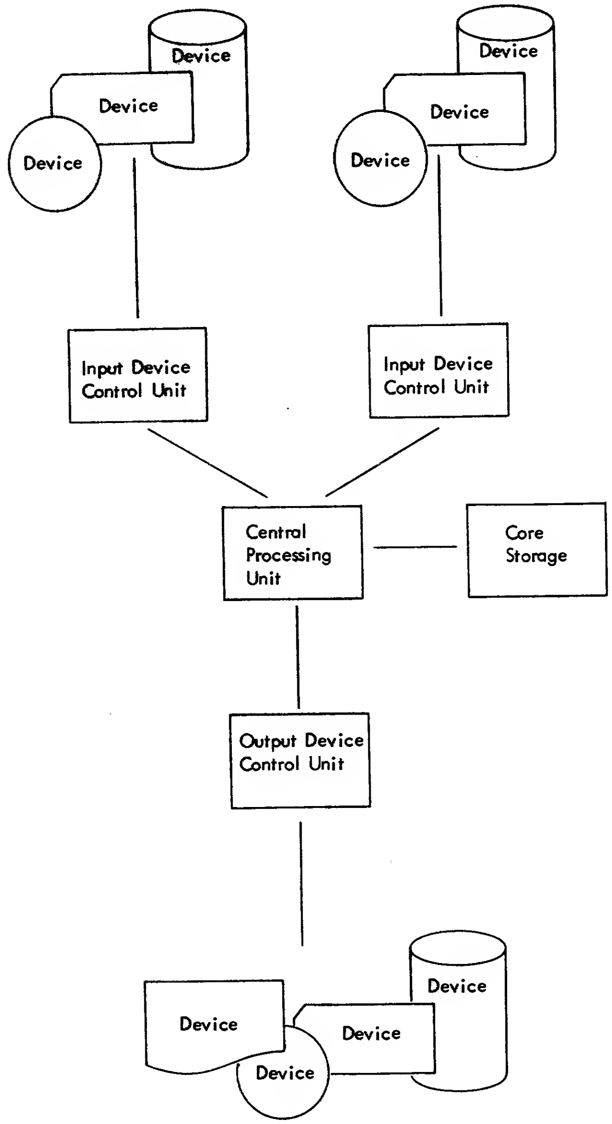


Figure 72. Input/output units in the data processing system

Usually, device operation is initiated by a program instruction that generates a command to an input/output channel. A control unit decodes the command and effects operation of the device.

Some control units control only a particular type of device, such as tape units. Others control different input/output devices; for example, the IBM 2841 control units can control the 2302, 2303, 2311, and 2321 DASD.

Input devices sense or read data from cards, magnetic tape, paper, magnetic ink characters, inscribed on paper documents, images on 35mm microfilm, or remote terminals via communication lines. The data is made available to the main storage of the system for processing. Output devices record or write information from main storage on cards, magnetic tape, and paper tape, prepare printed copy, produce microfilm images, make graphic displays, or transmit information over teleprocessing network.

Reading takes place as the input medium physically moves through an input device. Information is sensed or read and is converted to a code used within the computer system. The information is then transmitted to main storage.

Writing involves transferring data from primary storage to an output device. The computer code is made compatible with the output medium.

Most input/output devices are automatic; once started, they continue to operate as directed by the stored program until the entire file is processed. Instructions in the program select the required device, direct it to read or to write, and indicate the storage location into which data will be entered or from which data will be taken.

Some I/O devices are used for manual entry, and no medium for recording data is involved. Instead, data is entered directly into storage using a keyboard or switches. Locally, these devices may be a console keyboard, local terminals (such as the IBM 2740s), or graphic display terminals. Remotely, many types of teleprocessing terminals may be used. Instead of a recording medium, these terminals may require some amount of internal storage for holding (and perhaps analyzing) signals until a short message is completed, or until the terminals are polled (requested to transmit) and selected for data transfer.

Control Units

The type of information buffering required to coordinate the operations of the input/output device with the central processing unit (sometimes through transmission hookups) is one of the functions of the control unit. Other common functions are checking, coding, and decoding. If several similar devices are operating through one control unit, two principal functions are (1) determining priority of servicing, and (2) signaling device identification when requesting service for the input devices.

Conversely, on the way out, the control unit directs the data to the addressed output unit.

In some data processing systems, the traffic routing function of the control unit is referred to as orders. In such systems, the orders, as written by the programmer, consist solely of the address of the input/output device that the control unit must prepare to read or to write. In System/360, orders are part of a control command, relayed through a channel to the input/output control unit; they instruct the control unit to have a device perform a specified auxiliary operation – one that doesn't move data, such as rewind or seek.

Channels

Whereas the control unit is either included under the cover of an input/output device or located very close to a group of such devices, the channel (or channels) is contained within the central processing unit or is a separate piece of equipment near the CPU. It might be thought of as the computer's control unit for one or more input/output control units. It is almost a separate, small CPU devoted exclusively to managing the input/output control units and devices assigned to it. After the channel has once been activated by an initializing instruction from a program being executed in the CPU, it carries out one or more commands that are similar to a section (subroutine) of a program, but the important difference is that of overlapping operations. The program in the CPU can be continuing with other jobs while the channel is carrying out its own program of bringing data into or out of the main storage. Sometimes it is interleaving input and output in a seemingly simultaneous fashion, working with several input/output control units at once, and maintaining the proper destinations for the messages – whether they be storage allocation (for input) or control unit and device (for output).

The steps in a program in the CPU are called instructions; the steps in a program for a channel are called commands. Each command has an operation code that tells the channel what to do (for instance: read, write, control, sense, etc.); if it is a command that involves a data transfer, the command also has an address telling where to get or where to put the data in the storage system of the computer; if it is a control command that does not involve a data transfer, either it contains the order to be passed on to the control unit, or (in some computer systems) it contains the address of a location in storage where the order is located.

Just as the CPU is free to continue with its program once it has given an instruction to start a channel on its independent program of commands, so a channel is free to step through other commands (probably starting or terminating some other input/output transfer of data) as soon as it has commanded the control unit what to do and given it an order specifying the particular device. Thus, a channel is an intermediary input/output device that is constantly juggling the various input/output operations to make the most efficient use of time, not only by overlapping different inputs and outputs but by doing so without tying up the CPU.

As soon as a particular input/output transaction is completed, the device control unit signals the channel, which, in turn, signals the CPU with an "interrupt", meaning: "My particular job is done. As soon as convenient, use the data I have given you (if it was an input operation) and give me another command."

This idea of automatic interrupts (built into the design of the data processing system components), combined with careful preprogrammed commands to the channels and orders to the control units, leads to a far greater total amount of data handling per unit of time (sometimes described as throughput) than used to be possible.

Validity Checks

All data transferred between the input/output units and storage is automatically checked for validity. First, data is checked before being sent by the input device and checked when received by the output device. Second, certain data checks are also made within the central processing unit as it receives or sends data. These checks do not detect the use of wrong data; for example, if a 5 is entered instead of a 4, this error cannot be detected. However, if the indicated number or character is represented or coded incorrectly on the medium or within the machine, this is automatically detected.

Indicators, Keys, and Switches

All input/output units have indicator lights as well as operating keys and switches (Figure 73). The indicator lights show the status of a unit: on, off, ready, selected, and so on. The operating keys and switches are used primarily to start and to stop operations manually. The specific functions and use of the indicators, keys, and switches are described in the IBM manuals for particular machines and systems.

Control Panel

Some input/output devices are equipped with a control panel, which provides a means of editing, re-

Figure 73. Operator panel

arranging, deleting, and selecting data flowing through the device.

Basically, the control panel is similar to a telephone exchange switchboard. An incoming call lights a signal lamp that indicates to the operator the line on which the call is coming. After the call is answered, the operator plugs a cord into a hub that is internally connected on the board to the desired extension. Actually, the operator has completed an electrical circuit to give the correct result.

The control panel in a machine completes circuits internally by means of wires placed in the panel. The actual connections in the panel are made through holes called exit hubs and entry hubs. An exit hub is one that emits an impulse; an entry hub is one that accepts an impulse. The exit and entry hubs used depend on the functions to be accomplished. The panel is prewired by the operator for a specific job before it is placed in the machine.

Input units with control panels alter or change data after the data is read by the unit but before the data is sent to the main storage of the system. Output units alter or change data after the data is received from storage but before the data is punched or printed.

Control panels are removable and can be readily changed for different procedures, or a separate panel may be used for each operation.

Card Readers

Card reading devices introduce IBM punched card data into the computer system. The card reader moves or feeds cards past a reading unit that converts the data on the card into an electronic form. Two types of reading units are used: reading brushes or photoelectric cells.

In the brush type reader, cards are mechanically moved from a card hopper, through the card feed unit, and under reading brushes. The reading brushes electrically sense the presence or absence of holes in each column of the card (Figure 74).

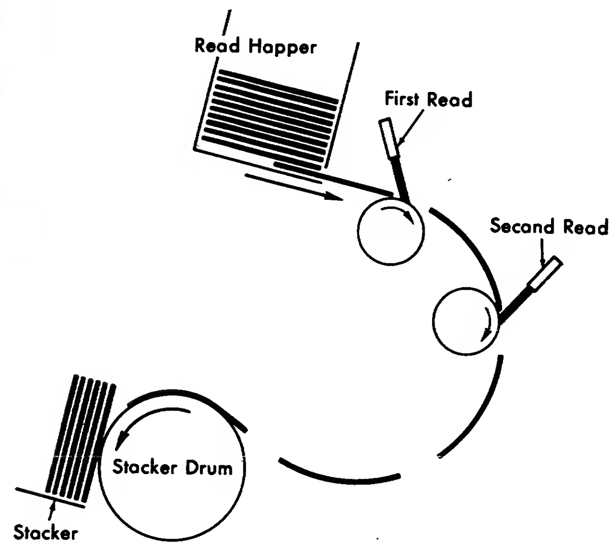


Figure 74. Read feed

This electric sensing converts the information of the card to electrical impulses that can be detected by the card reader circuitry and stored as data. After cards are read, they are moved from the card feed unit and placed in the card stacker in the same sequence in which they were fed into the reader. Some readers have two sets of reading brushes. As a check on the validity of the reading process, each card can be read twice as it moves through the card feed unit.

The photoelectric type of card reader performs the same functions as the brush type; the difference is in the method of sensing the holes. Photoelectric cells are activated by the presence of light. As the punched card is passed over a light source in the card reader, light passing through the punched holes activates photoelectric cells, one cell for each column of the card.

Card reading speeds vary from about 12 cards to 1000 cards a minute, depending on the type of card reader.

Card Punches

Output from the computing system is recorded in cards by a card punching device. The card punch automatically moves blank cards, one at a time, from the card hopper, under a punching mechanism that punches data received from storage (Figure 75). After

the card is punched, it is moved to a checking station, where the data is read and checked with the information received at the punching station. The card is then moved to the stacker.

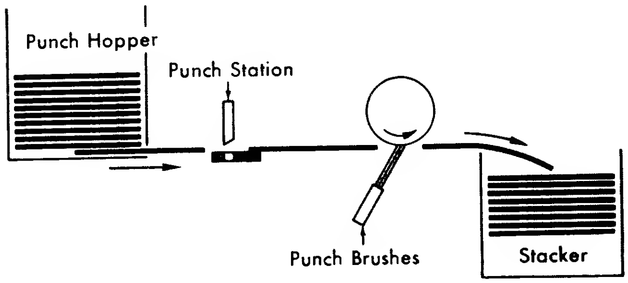


Figure 75. Punch feed

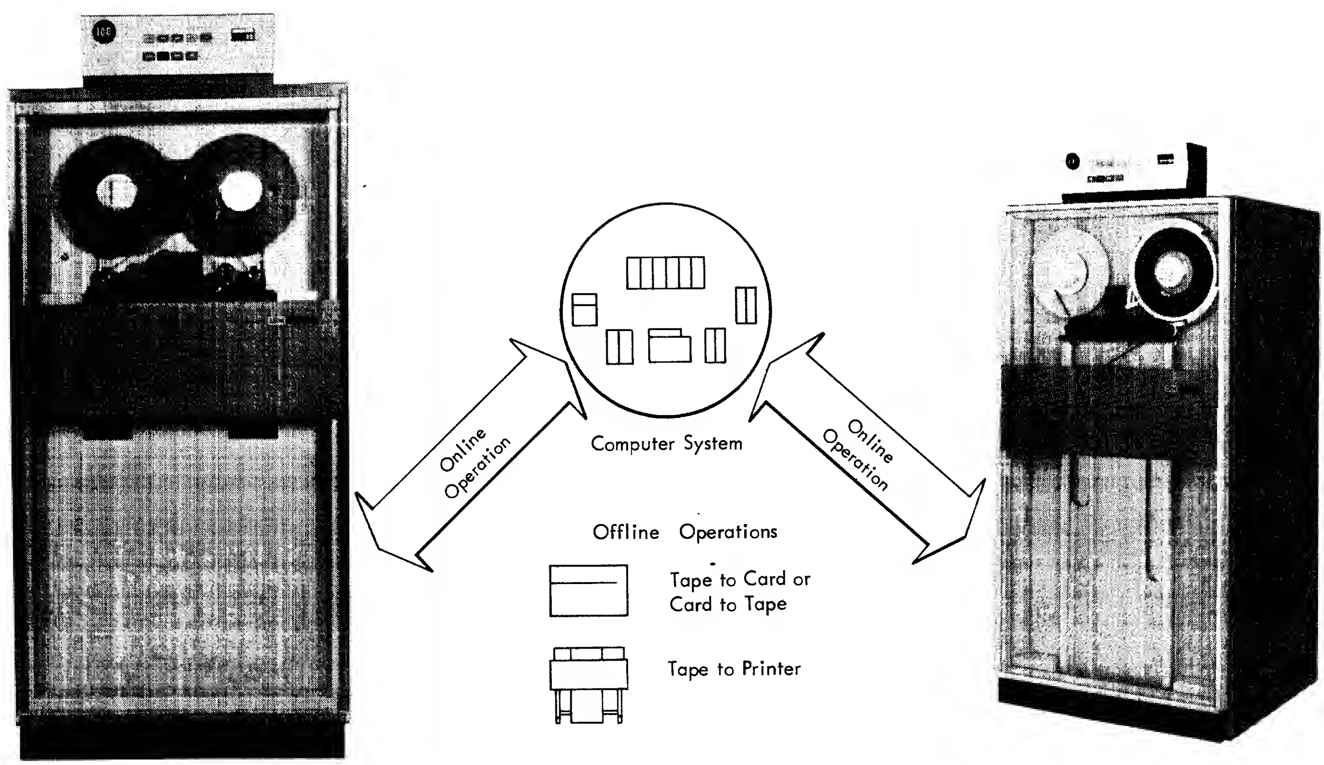
Card punching speeds vary from about 12 to 500 cards per minute, depending on the type of card punch.

Magnetic Tape Units

Increasing internal speeds of computers demand high-speed input and output devices so that system operations are not held back waiting for input or by an inability to get processed data out of the computer. Magnetic tape units, with their dual capability of input and output, have provided continued increases in the speed of data transmission to and from the computer and, at the same time, have provided increased data storage capacity per reel of tape.

All magnetic tape units are basically similar in operation, but design improvements have brought about functional differences, increased tape applications, and easier operation (see Figures 76 and 77).

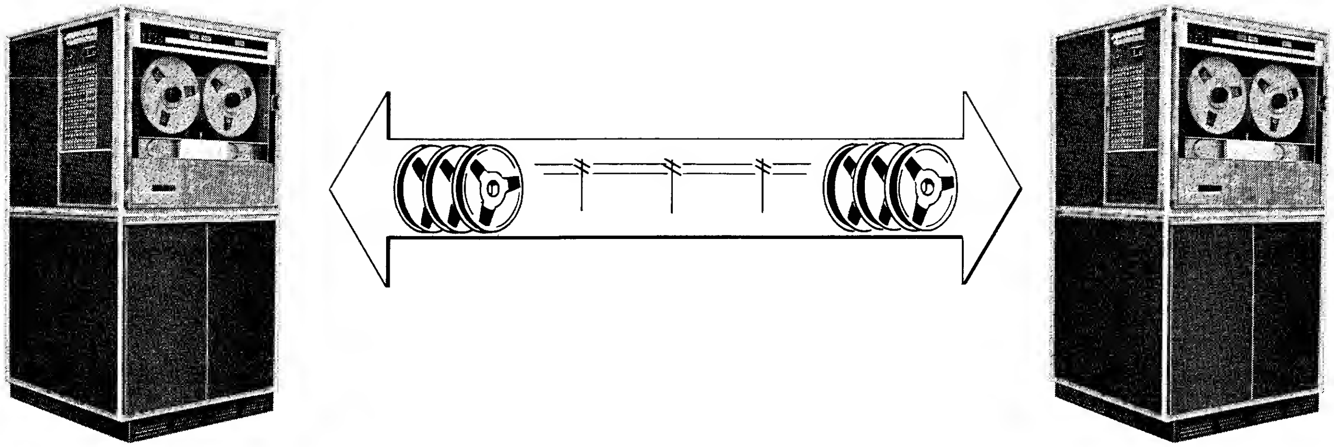
The IBM 2400 series units transport tape past the recording head in a continuous movement and at a constant speed. The tape is always in motion during reading and writing. Tape moves at 37.5, 75, 112.5, or 200 inches per second, depending on the tape unit.



IBM 2401 Tape Units are manually loaded with tape and may be used online (attached the computer system) or offline.

IBM 2420 Model 7 Tape Units use a wrap-around cartridge for standard 10½-inch reels that threads and rewinds automatically.

Figure 76. IBM magnetic tape units



The IBM 7702 Magnetic Tape Transmission Terminals transmit magnetic tape data between remote locations.

Figure 77. IBM tape transmission terminals

A full reel of ½-inch-wide tape holds 2400 feet of tape, weighs about four pounds, and can contain data equivalent to about 400,000 fully punched cards.

Loading Tape Units

Before the tape unit can read or write, it must be prepared for operation. This preparation involves loading two tape reels (except cartridges) on the tape unit and threading the magnetic tape through the tape transport mechanism. For ease of threading, the head assembly separates to accept tape; it is then closed to bring the magnetic tape into close contact with the read/write head for reading and writing. The operation is similar to threading a home movie projector (Figure 78).

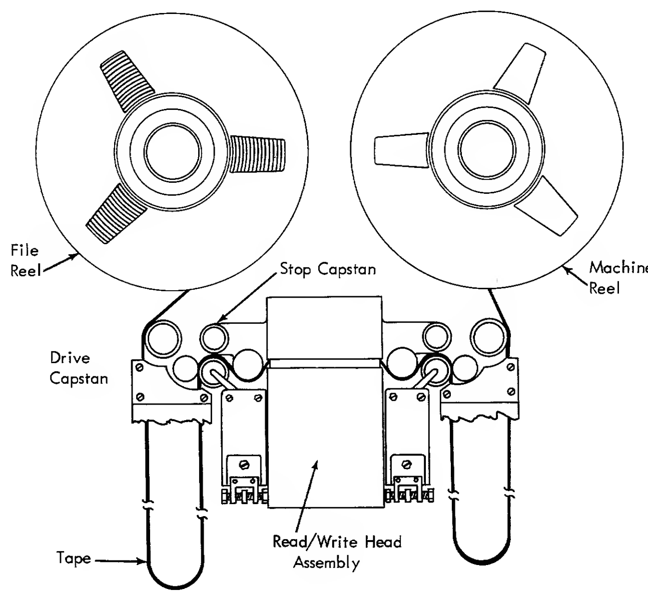
During operation, tape moves from the file reel through the left vacuum column across the read/write head, through the right vacuum column to the machine reel. The loop in each vacuum column acts as a buffer to prevent high-speed starts and stops from breaking the tape. Vertical vacuum columns are used in some units; horizontal columns are used in the incremental and variable-speed machines.

Vacuum-actuated switches in the columns control magnetic clutches that permit the two reels to rotate independently. The file reel feeds tape when the loop reaches a minimum reserve length in the left vacuum column, and the machine reel winds tape when the loop reaches a point near the bottom of the right vacuum column.

Tape may be rewound or backspaced to the beginning of the reel. Rewind speeds are as high as 500 inches per second.

Loading Tape Reel Cartridges

Loading of cartridges is accomplished automatically after the dust-resistant cartridge is placed in the drive by the operator; the time-consuming manual threading operations involved in loading tape are eliminated (Figure 79).



During reading or writing on all tape drives other than Hypertape, tape is transferred from the file reel, past the read/write head, to the machine reel. During backspacing or rewinding, tape movement is from the machine reel to the file reel.

Figure 78. Tape feed schematic

Figure 79. Tape Reel cartridge operation

Reading From and Writing on Magnetic Tape

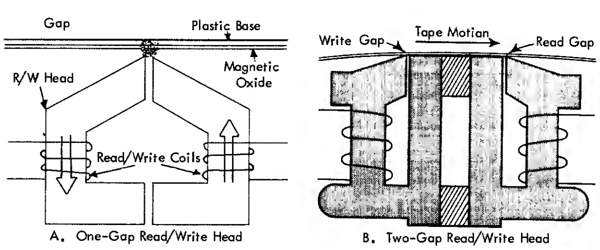
The magnetic tape unit reads or writes data as tape moves past the read/write head. Two types of heads are used in present IBM magnetic tape units, but the general principles of writing and reading tape are the same for both types (see Figure 80).

Writing on magnetic tape is destructive; that is, as new information is written, old information is destroyed. Reading is nondestructive; the same information can be read again and again.

Information is written on tape by magnetizing areas in parallel tracks along the length of the tape.

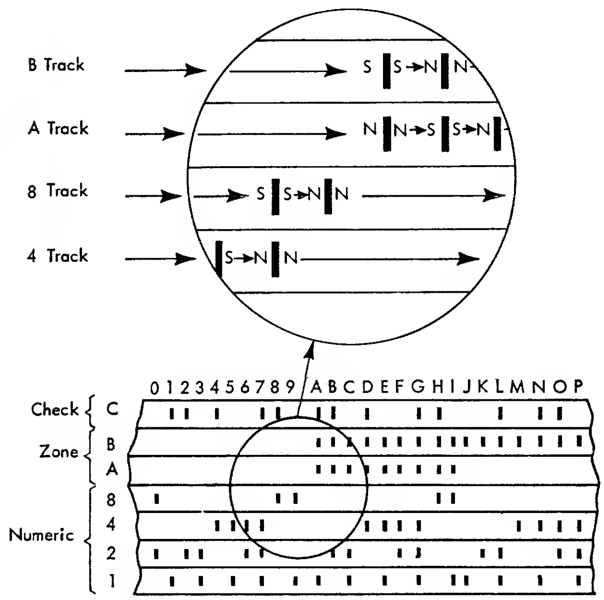
There is one write coil in the write head for each recording track. Electrical current flowing through the coils magnetizes the iron oxide coating of the moving tape and erases previously written information (see Figure 81).

In incremental tape units, the tape actually is motionless during writing, but the size of the recorded bits is almost the same as in the other tape units. Data



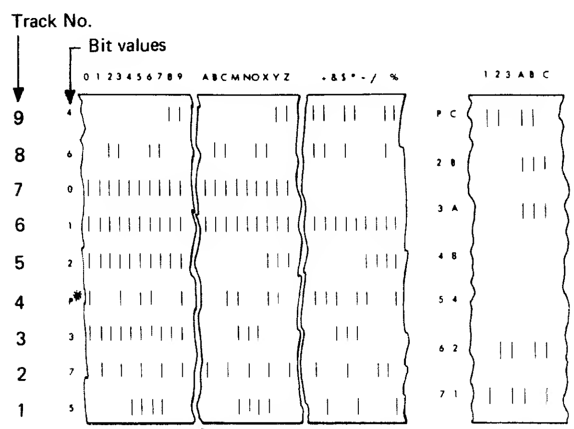
In the one-gap read/write head, reading and writing take place at the same gap. In the two-gap head, writing occurs at one gap, and reading occurs at the other. The two-gap head offers advantages discussed in the tape validity-checking sections.

Figure 80. Magnetic tape unit read/write heads



New data is written on magnetic tape by changing the direction of current flow and magnetic polarity from north (N) to south (S) in some of the write coils. This causes a change in the affected tracks. The coded pattern of 0 and 1 bits across the width of the tape represents data received from the computer.

A. Magnetic recording of seven-track BCD code on tape



*The P bit position produces odd parity.

B. Nine-track (EBCDIC) and seven-track tape data format comparison

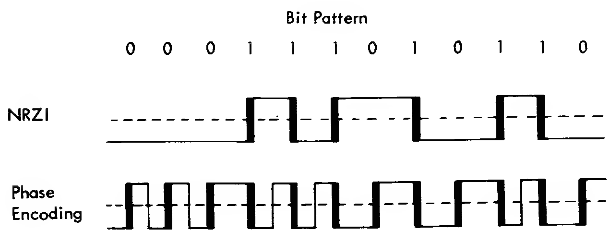
Figure 81. Data recording on magnetic tape

generated on the incremental and variable-speed tape units is usable on other tape units adapted to seven-track data.

Checking Magnetic Tape Data

Data recorded on magnetic tape must be accurate so that errors are not sent through the system. Data is therefore checked to ensure that valid characters are recorded and to verify that the recorded bits are of effective magnetic strength.

Two methods of recording are used on IBM magnetic tape. The phase encoding method is used on IBM 2400 Series Magnetic Tape Units; other magnetic tape units use the Non-Return-to-Zero-IBM (NRZI) method (see Figure 82).



In the NRZI method of recording, a change in magnetic flux is interpreted as a 1 bit; lack of a flux change is interpreted as a 0 bit. The phase-encoding method of recording used on Hypertape results in a continuous wave pattern, even when a record contains all zeros.

Figure 82. Comparison of NRZI and phase-encoded bit patterns

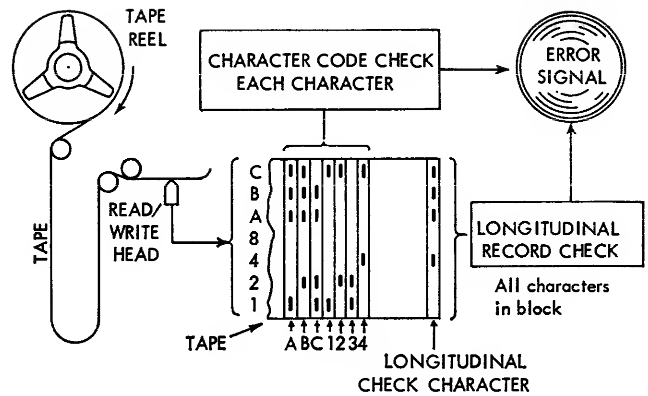
The NRZI method of data recording is very reliable, but it has given way to the new phase encoding method because of the increased densities of recording possible on the 2400 series units.

The tape error detection system used on NRZI tapes uses the principle of simple parity checking. With this system, it is possible to detect virtually all tape reading and writing errors (Figure 83).

Simple parity checking indicates the error, but not the kind of error. Similar double-bit errors in two characters of a record could conceivably cancel each other and indicate correct parity. However, this coincidence is extremely rare.

With the 2420 Model 7, single track errors are corrected in flight without impairing tape performance. As a result, most corrections are made without interrupting processing.

Tape units with two-gap heads provide increased checking while writing. Tape being written passes first over the write gap (to record data) and then over the read gap; the information that has been written is automatically read and checked.



Information read from tape is checked two ways. A character code check (vertical check) is made on each column of information to ensure that an even number of bits exists for each character read. If an odd number of bits is detected for any character or column of bits, an error is indicated, unless the computer operates in odd parity. A longitudinal record check is made by developing an odd or even indication of the number of bits read in each of the seven bit tracks of the record, including the bits of the check character. If any bit track of the record block indicates an odd number of bits after it is read, an error is indicated, unless odd parity is required by system design.

Figure 83. Seven-track validity checks, BCD mode, even parity

When an error occurs during the writing operation, it is detected at the read gap, and an error indication is made. Programming must test the indicator and take an appropriate corrective action. The machine does not stop with the error section of tape positioned over the read gap; tape motion continues past the end of the record block. Then the machine may be instructed to backspace the tape and rewrite, again checking for an error.

Figure 84. Wrap-around cartridge

Tape Records, Interblock Gap, and Tapemark

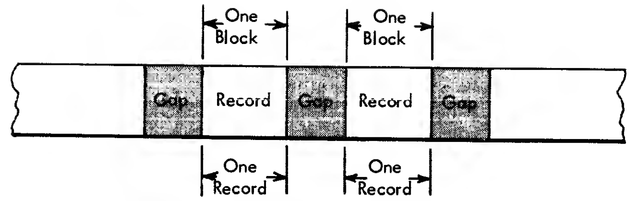
Records on tape are not restricted to any fixed length of characters, fields, words, or blocks. Records may be any practical size within the limits of internal storage capacity.

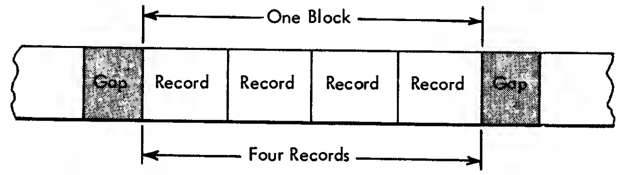
Blocks of records, including blocks consisting of a single record, are separated on tape by an interblock gap, a length of blank tape about .6 inch on 2400 series tape units, and .75 inch on all other tape units. During writing, the gap is automatically produced at the end of each block of records. During reading, the block begins with the first character sensed after a gap and continues without interruption until the next gap is reached. The interblock gap also allows time for starting and stopping the tape between record blocks.

A tapemark (a special character represented in hexadecimal as 7F; see Figure 39) indicates the end of a file of records (see Figure 85). Most computers write and read tapemarks.

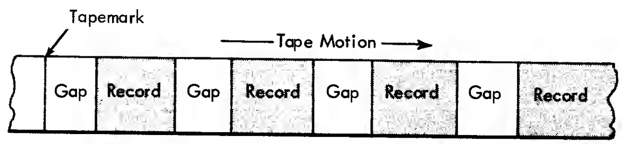
Tape Unit Characteristics

Among tape units, the major performance considerations are the speed at which tape is moved across the read/write head and the recording density of information on tape. These two factors determine important characteristics of character rate, tape access time (sometimes called gap time), and character time. Some other differences among tape units involve the length of the interblock gap, the extent and method of checking the validity of recorded data, and provisions for protecting recorded data. Figure 86 shows the major differences among tape units.





On magnetic tape, a single unit or block of information is marked by an interblock gap before and after the data. A record block may contain one record or several.



The interblock gap followed by a unique character record is used to mark the end of a file of information. The unique character, a tapemark, is generated in response to an instruction and is written on the tape following the last record of the file.

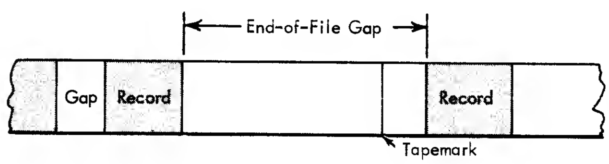


Figure 85. End-of-block and end-of-file indications on tape

	2401			2415						2420
	Model 4	Model 5	Model 6	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
Bytes per second	60,000	120,000	180,000	15,000	15,000	15,000	30,000	30,000	30,000	320,000
Density (bytes per inch)	1,600	1,600	1,600	800	800	800	1600	1600	1600	1,600
Tape speed (inches per second)	37.5	75.0	112.5	18.75	18.75	18.75	18.75	18.75	18.75	200
Nominal Interrecord gap (inches)	.6	.6	.6	.6	.6	.6	.6	.6	.6	.6
Nominal IRG time (milliseconds)	16.0	8.0	5.3	32	32	32	32	32	32	3.0
Rewind time, including reload (minutes)	3.0	1.4	1.0	4.0	4.0	4.0	4.0	4.0	4.0	1.0
Rewind and unload time (minutes)	2.2	1.5	1.1	4.0	4.0	4.0	4.0	4.0	4.0	1.1

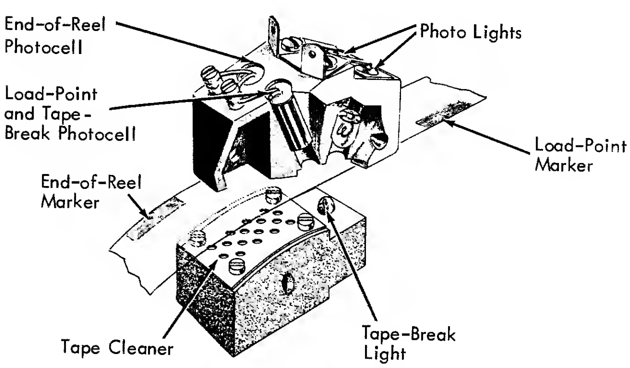
Figure 86. Characteristics of IBM 2400-series magnetic tape units

Maximum and Effective Character Rates

Because an interblock gap is placed between each record or block of records on tape, the total time required to read a record must include time to space over the gap; this is called access time to the data. Access time is given (Figure 86) for each tape unit on the basis of tape speed and length of interblock gap. Access time must be considered when determining the actual or effective character rate of a tape unit.

Load-Point and End-of-Reel Markers

Magnetic tape must have some blank space at the beginning and end of the reel to allow threading through the feed mechanism of the tape unit. Markers, called reflective strips, are placed on the tape to enable the magnetic tape unit to sense the beginning and the end of the usable portion of tape. Photoelectric cells in the tape units sense the markers as either the load-point marker (where reading or writing is to begin) or the end-of-reel marker (where writing is to stop). The tape unit does not recognize the end-of-reel marker when reading tape; a tapemark, written on the tape, signals an end-of-reel condition (see Figure 87).

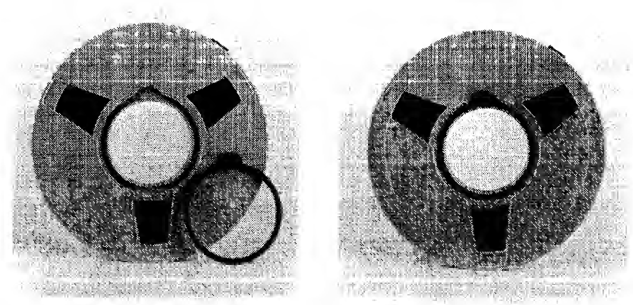


Tape markers are small pieces of transparent plastic with a thin, vapor-deposited film of aluminum on one side. Pressure-sensitive adhesive covers the aluminum film. The markers are fastened manually to the base (uncoated) side of the tape. New reels of IBM tape have the markers in position. Photocells in the tape unit sense the markers as they pass; broken tape is also detected.

Figure 87. Photosensing markers

File Protection

Because writing automatically destroys any previous information on the tape, a file protection device is used to prevent accidental erasure of information. This



On noncartridge tape, the file protection device is a plastic ring that fits into a round groove molded in the tape reel. When the ring is in place, either reading or writing can occur. When the ring is removed, writing is suppressed and only reading can take place; thus, the file is protected from accidental erasure.

Figure 88. File protection devices

device is used when tapes are to be saved for further reference (see Figure 88).

Direct Access Storage Devices

These have been discussed in the section entitled "Storage Devices". However, DASD's may be thought of as input/output devices.

Paper Tape Reader

The paper tape reader shown in Figure 89 reads data represented as punched holes in five, six, seven, or eight-channel paper tape at a rate of up to 1000 characters per second. As it moves or feeds the tape past a reading unit, the presence or absence of holes in the tape is sensed and converted to electronic impulses that are used as data by the computer system. Accuracy of reading is determined by making a parity check (where characters are written with parity, as in eight-channel code). The speed of reading, from 150 to 1000 characters per second, depends on the type of reader and the lengths of the records.

For faster paper tape input to the computer system, the data to be converted may be written on magnetic tape in an offline operation at 150 paper tape characters per second (Figure 90). The recorded tape may then be placed on a magnetic tape unit and read into

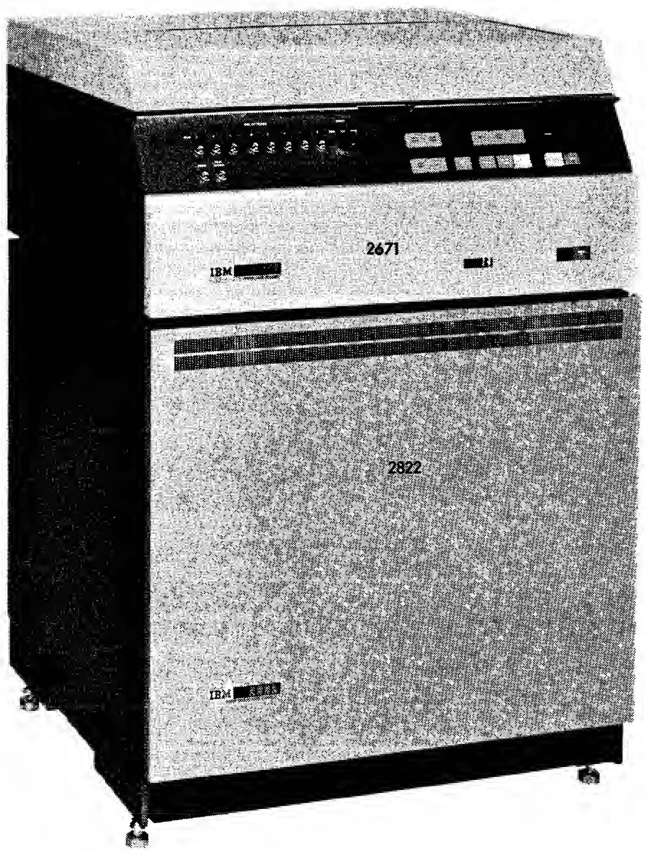


Figure 89. IBM 2671 Paper Tape Reader and 2822 Paper Tape Reader Control

the computer system at the much higher magnetic tape input rates (see table in Figure 86).

Paper Tape Punch

Data from the computer system is recorded as punched holes in paper tape by an automatic tape punch. Data received from main storage is converted to a tape code and punched in blank tape as the tape is moved through a punching mechanism. Accuracy of data recorded is verified by a parity check for each

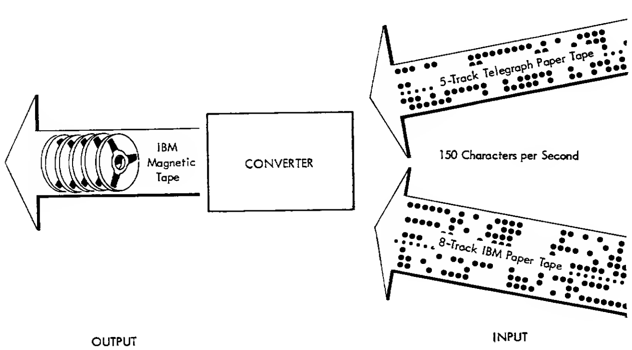


Figure 90. Data conversion-paper tape to magnetic tape

character (eight-channel code, for instance). Tape is punched at a density of ten characters to the inch and at rates of 15 and 150 characters per second.

Printers

IBM printing devices provide a permanent visual record of data from the computer system. Speeds of printing vary from 10 to 2400 characters per second.

As an output unit, the printer receives data, symbolized in electronic form, from the computer system. The electronic symbols enter appropriate circuitry and cause printing elements to be actuated. All printing devices have a paper transport that automatically moves the paper as printing progresses.

The major printing devices consist of the print wheel printer, wire matrix printer, chain printer, incremental bar printer, and the typewriter.

The print wheel printer is equipped with 120 rotary print wheels (Figure 91), each of which has 48 characters of type, including numerals, alphabetic symbols, and special characters. At the time of printing, all 120 print wheels are correctly positioned to represent the data to be printed. Printing occurs as a complete line of 120 characters, at a speed of 150 lines per minute.

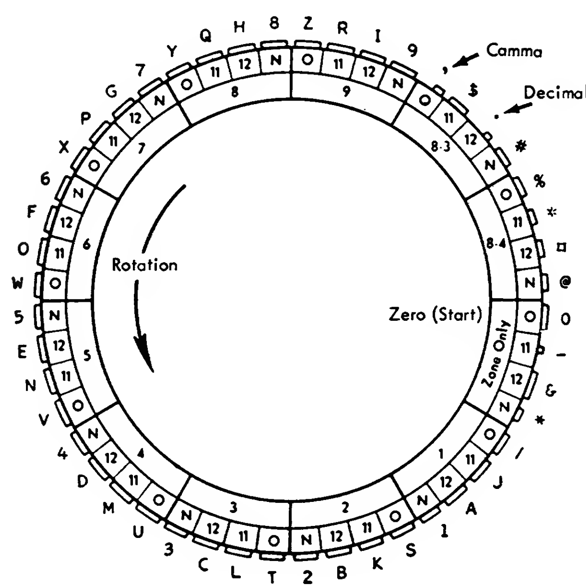


Figure 91. Print wheel

In the wire matrix printer, each character is printed as a pattern of dots, formed by the ends of small wires, arranged in a five-by-seven rectangle (Figure 92). By extending selected wires, the patterns may be arranged in the shape of 47 different characters, including all letters of the alphabet, the digits 0 to 9, and eleven special characters used for punctuation and report printing (Figure 93). Selected wires are pressed

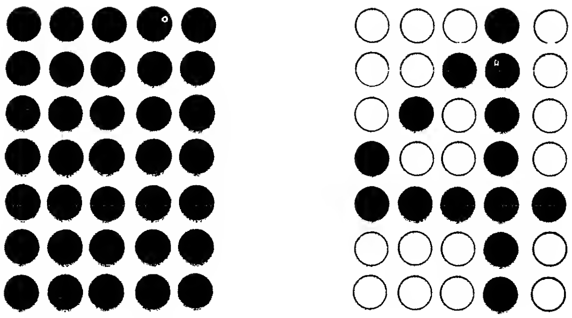


Figure 92. 5 x 7 dot pattern

against an inked fabric ribbon to print the characters on paper. Characters are printed 120 to the line, at a rate of 500 or 1000 lines per minute, depending upon the device model.

The chain printer is an electromechanical line printer using engraved type. Alphabetic, numeric, and special characters are assembled in a chain (Figure 94). As the chain travels horizontally, each character is printed as it is positioned opposite a magnetically actuated hammer that presses the paper against one piece of type in the moving chain. Up to 132 positions may be printed on one line, at speeds of up to 1285 lines per minute. The print chain can be easily changed to provide a choice of print fonts.

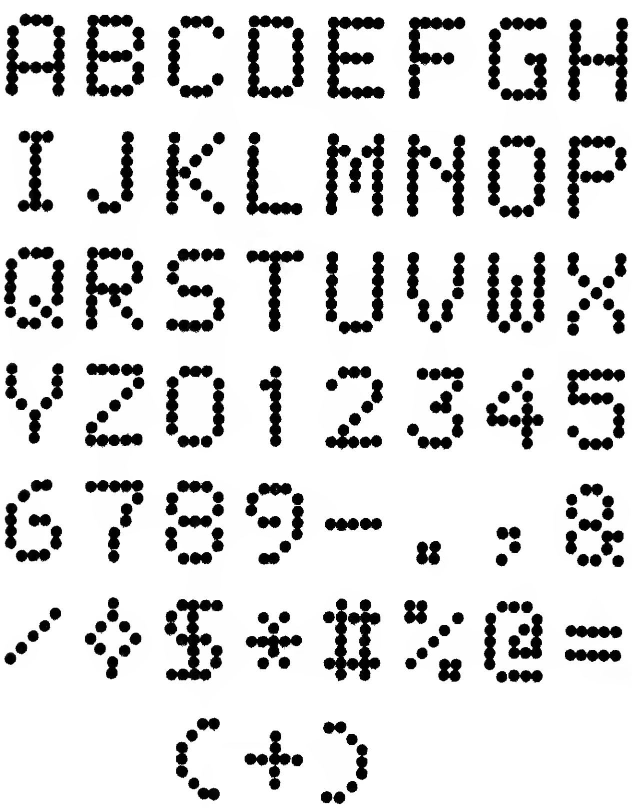


Figure 93. Wire printing dot pattern

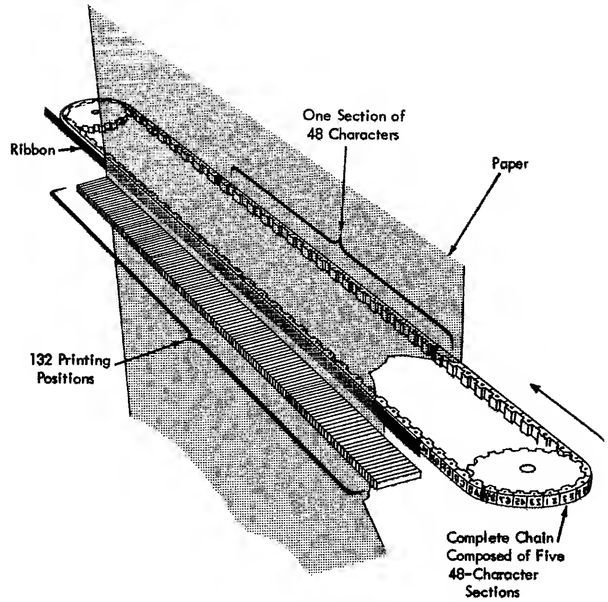


Figure 94. Print chain

The incremental bar printer has a bar, containing the print characters, which travels back and forth in a horizontal plane. To print, a magnet releases a spring-loaded hammer at the proper time so that the desired character is pressed against the ribbon and paper. Such printers can achieve a print speed up to 240 characters per minute.

The typewriter that is used as an output device (Figure 95) is similar to the one used manually. The major difference is that control of the typewriter and the printing occurs automatically as directed by the

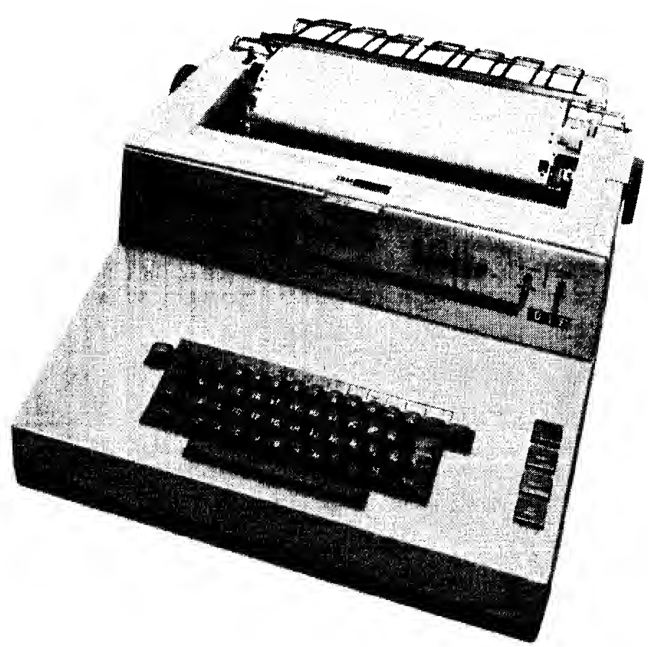


Figure 95. 1052 Printer-keyboard

stored program. Printing speed is about 600 characters per minute; spacing and carriage return are automatic. With the 1052 printer-keyboard (using the "golf ball" printer), printing speed is almost 900 characters per minute.

Universal Character Set

A special feature available for S/360 printers is the universal character set (UCS), which enables a user, with a customized print chain, to obtain maximum printer efficiency. If needed, a user can have an expanded character set (up to 240 different characters on any one chain) that is capable of using any of the 256 EBCDIC codes, except "null" or blank (hex 00 and 40).

A user can customize a print chain by having only those characters, symbols, etc., necessary for a particular job. (IBM currently has a variety of print chains available that meet most needs.) The user then assigns a code to represent each character on the chain. The codes are entered into a buffer (located on the control unit) in the positions that the characters appear on the print chain (there are 240 buffer positions and 240 chain positions). When the user is printing, the printer prints the character that corresponds to the code in the buffer position.

On printers without the UCS feature, there is a fixed print cycle (that is, a constant time is needed to print a line). The UCS user has a variable and (in most cases) a faster print cycle, which ends when the last character on the line is found.

Character Recognition Input Units

With the advent of high-speed automatic data processing systems, there came a realization that wide use could be made of input units capable of reading data that could also be read by people. The processing of over 13 billion checks yearly and the inestimable volume of other notices – insurance billings, magazine subscription renewals, invoices, manufacturing routing slips, utility bills, and so on – could be greatly speeded by the use of man-machine recognizable characters. Two systems that accomplish this are magnetic characters and optically readable characters.

Magnetic Character Readers

As a specialized means of input to computer systems, IBM magnetic character readers provide banks with a time-saving method of reading and processing large volumes of daily transactions. These machines read card and paper documents inscribed with the E13B type font approved by the American Bankers Association. A second important labor-saving feature of the magnetic character readers is their ability to sort the magnetically inscribed documents in offline operations (see Figure 96).

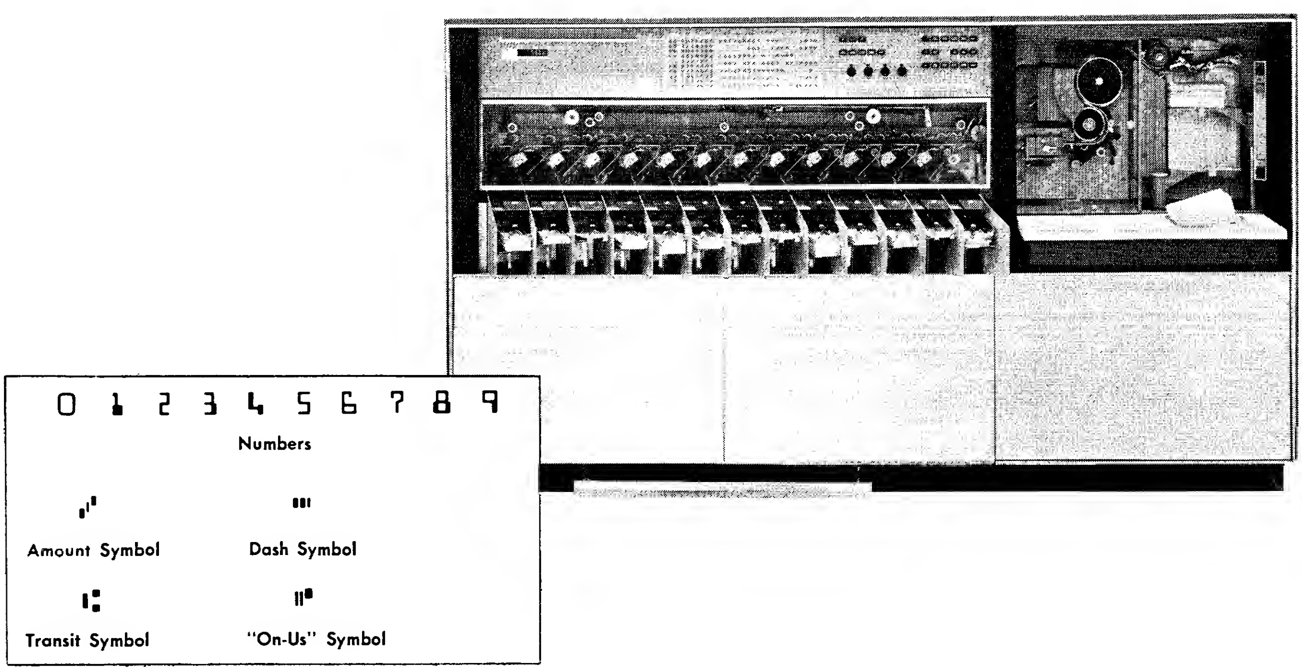


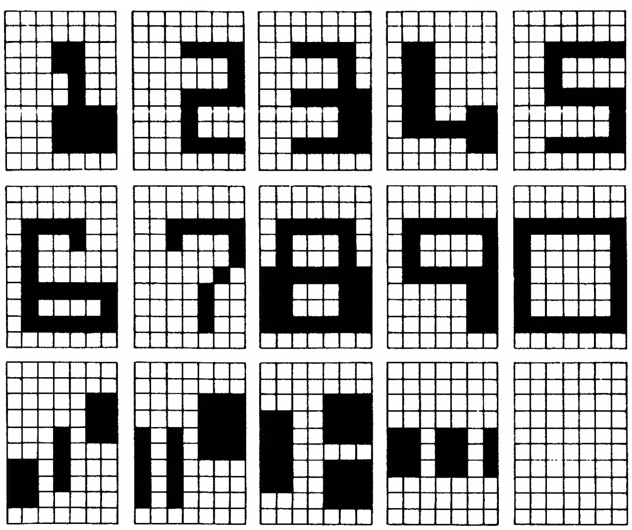
Figure 96. IBM 1419 Magnetic Character Reader and magnetic ink characters

The reader examines the shape of each magnetic ink character passing under the read head, and ten data channels send signals to an electronic storage device called the character matrix. The matrix has a storage location for each of 70 character segments, and, as documents pass under the read head, lack of any appreciable signal from a character area segment causes the machine to store a 0 bit in that storage location. The presence of a significant signal (indicating that magnetic ink is under the reading gap) causes the machine to store a 1 bit in the specified storage location. The bit structure entering the matrix is also displayed in the character matrix lights of the indicator panel.

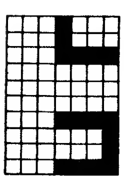
After the entire character area has passed under the read head and all segments have been read, a pattern of the character shape is in the character matrix as a configuration of 0 bits and 1 bits (see Figure 97). To verify the accuracy of processed data, the reader automatically checks each character as it is being read.

An exhaustive study of the 14 characters shown in Figure 97 has determined that thousands of 0 bit and 1 bit configurations can be considered acceptable patterns for each character, even when portions of the character are missing. All other patterns are considered invalid.

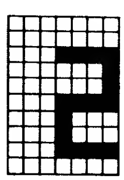
When the machine determines that a character is valid, the reader stores the character in another storage location called the character register. The character remains there until it is no longer needed for processing. If the machine determines that the pattern is invalid, the recognition circuits provide the machine with an error signal.



Each magnetic character passing the read head in the reader is sensed and examined. The reader looks for key recognition shapes and characteristics. If the character is slightly out of position as it passes under the read head, the signals sent to the matrix form what is called a folded character. The reader automatically unfolds the pattern by shifting it vertically to check for recognition.



Folded Character Pattern in Matrix



Character Pattern Shifted in Matrix

Figure 97. Matrix patterns of E13B characters

Optical Character Readers

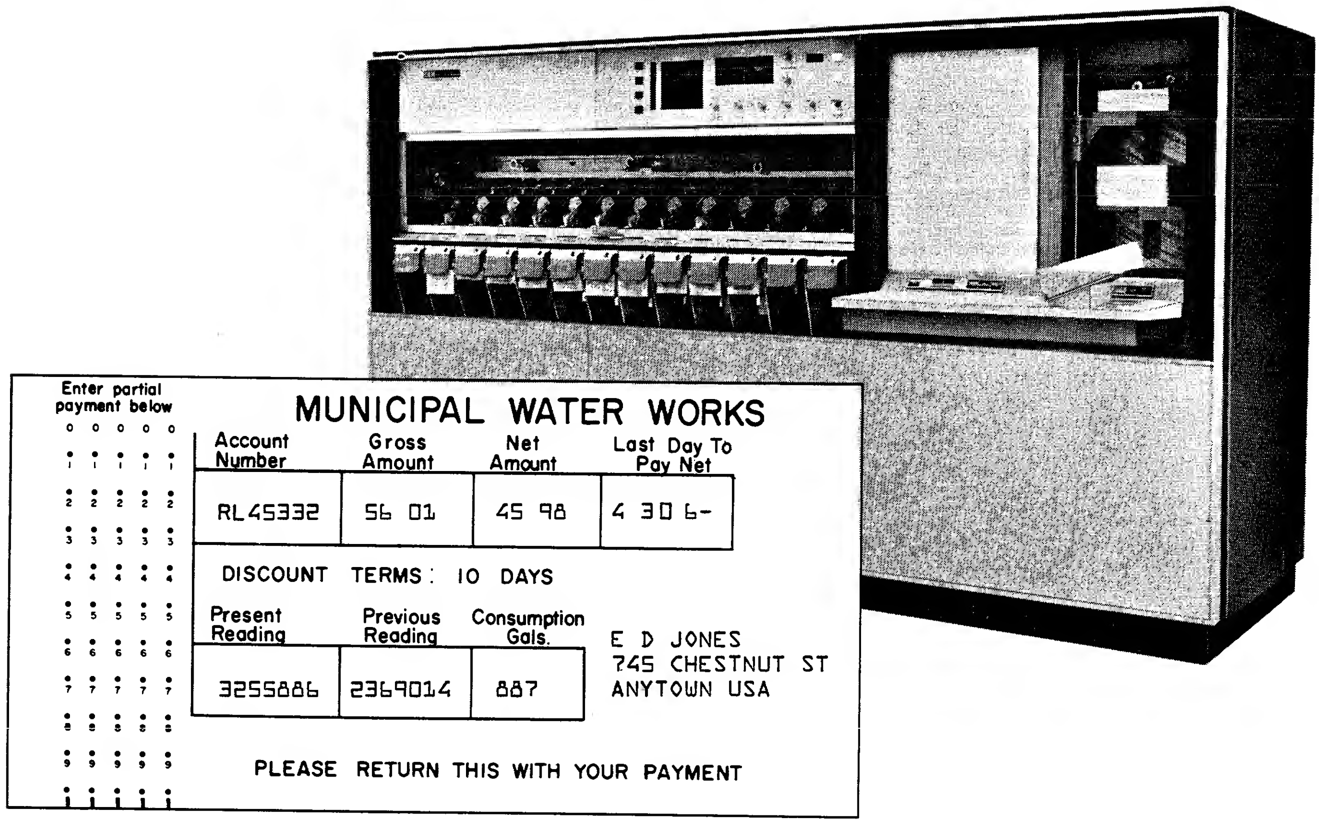
The optical character reader reads uppercase letters, numbers, and certain special characters from printed paper documents and introduces the data into a computer system. Transcribing of source data to cards or tape is eliminated, and the time between receipt of source documents and their entry into the data processing system is greatly reduced.

The principal operating action of the optical character reader is provided by a rotating drum that transports documents from a hopper past an optical scanning station. The scanner consists of a powerful light source and a lens system that distinguishes between black and white patterns of reflected light. These light patterns are read as a number of small dots and are converted into electrical impulses to develop a character pattern. When the pattern of the optically read character matches a character pattern in the reader's character recognition circuits, the character is recorded and transferred into the computer system for processing. The read and recognition operation is automatic and takes place at split-second speeds.

Optical readers can perform an additional operation known as mark-reading, the reading of ordinary pen or pencil markings. The mark, when placed in a specified location on the source document, represents specific information. This feature has many important applications, such as recording of partial payments directly on a customer's bill and immediate processing of payment, recording of meter reading cards at the customer's utility meter, and so on. (See Figure 98.)

Graphic Display Units

In some data processing systems, terminals produce a visual display of records in main or secondary storage, as called for by an operator. The IBM 2260 Display Station (Figure 99) is such a terminal, used with System/360. The information transfer to and from System/360 takes place at 2560 characters per second. Remote data transfers over telephone lines occur at transmission line speeds of 120 or 240 characters per


Enter partial payment below

MUNICIPAL WATER WORKS

Account Number	Gross Amount	Net Amount	Last Day To Pay Net
RL45332	56 01	45 98	4 30 6-

DISCOUNT TERMS: 10 DAYS

Present Reading	Previous Reading	Consumption Gals.
3255886	2369014	887

E D JONES
745 CHESTNUT ST
ANYTOWN USA

PLEASE RETURN THIS WITH YOUR PAYMENT

Figure 98. IBM 1428 Alphameric Optical Reader and mark-sensing document

Figure 99. IBM 2260 Display Station

second. Once written, the information is retained on the screen for as long as desired.

The viewing area (four inches by nine inches) has the capacity for six or twelve rows of 40 or 80 characters each, for a total of from 240 to 960 characters at a time, depending upon the model of display control used. Standard for display are all 36 alphameric characters, plus 25 special characters. Each character is designed on the basis of a five by seven matrix, with an appearance that is similar to that of a printer (Figure 92).

All input to the 2260 is through an alphameric or numeric keyboard. Input data typed on the keyboard goes into the buffer of the 2848 Display Control and is immediately displayed on the tube face for a visual accuracy check. Before the message is released to the host computer for action, the operator can backspace/erase/correct an entry, erase an entire input message and reenter it, or erase the entire display.

Under operator or program control, output information is written on a cleared display or added to an existing display on the tube face. Output is on the cathode ray tube screen, or it may be directed to a

1053 Printer, where it is printed out in the same format as displayed on the tube face.

Another display unit is the IBM 2250, described under "Visual Output" in the section entitled "Data Representation".

Consoles

The console of a data processing system (Figure 100) is used by the operator to control the system and monitor its operation. Using keys, switches, audible tone signals, and display lights on the console, the operator can:

1. Start and stop the computation.
2. Manually enter and extract (or display) information from internal storage.
3. Determine the status of internal electronic switches.
4. Determine the contents of certain internal registers.
5. Alter the mode of operation so that, when an unusual condition occurs, the computer will either stop or indicate the condition and proceed.
6. Change the selection of input/output devices.
7. Reset certain types of computers when error conditions cause them to halt.

In some large data processing systems, the main console is connected only to the central processing unit and may be augmented by separate consoles that are used for engineering functions and for additional input/output control.

Terminals

IBM currently has two types of terminals used in telecommunications. Because of the way they receive and send messages, they are called start/stop and synchronous transmit/receive (STR) terminals.

The start/stop type terminals include the IBM 1050, 1030, 1060, and 1070. The operation of these terminals is based upon asynchronous transmission (in which each information character is individually synchronized by a start character). In this type of transmission, each group of code elements corresponding to a character signal is preceded by a start signal, which serves to prepare the receiving mechanism for the reception and registration of a character. In addition, each character is followed by a stop signal, which serves to bring the receiving mechanism to rest in preparation for the reception of the next character.

The STR type terminals include the IBM 1009, 1013, 7702, 7711. In the STR mode of transmission,

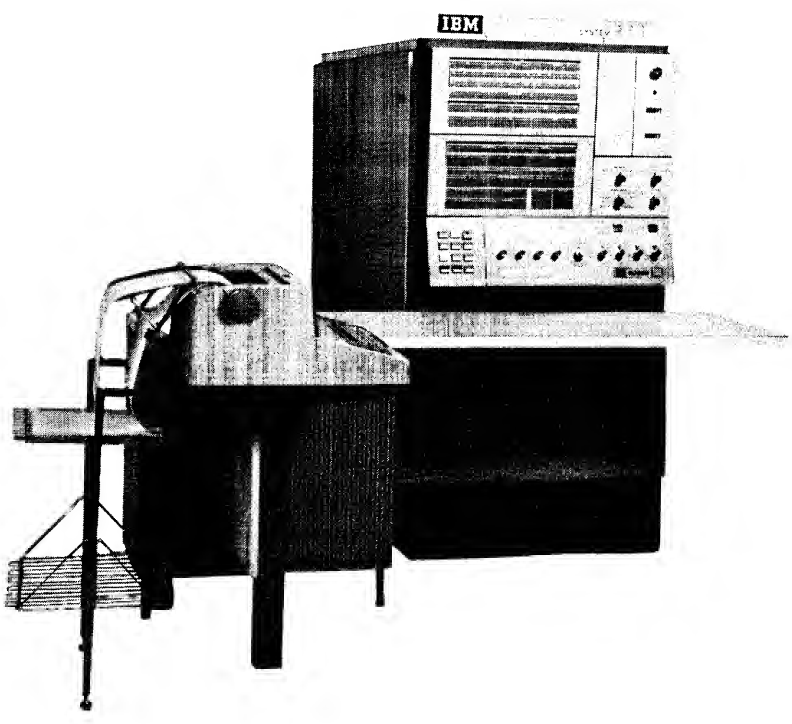


Figure 100. Console – IBM System/360 Model 30

the code bits of one character are immediately followed, without a start or stop signal, by the code bits of the next character.

The most common mode of communication between STR terminals is the half duplex mode (sometimes called "handshaking mode"). In this mode, two terminals are in constant communication with each other. The first terminal, if it has no message to send, sends special nondata characters (called IDLE characters) for about 1.5 seconds. The second terminal, meanwhile, is synchronized to receive this signal. After 1.5 seconds, the two terminals reverse mode (that is, the first terminal receives, and the second terminal sends the IDLE's). When one terminal has a message to send, it sends a special code that "asks" permission of the other terminal to do so. When permission is granted, the terminal sends the message (wherein each code bit is part of the characters of the record). When the terminal is finished sending the record, it sends an end-of-record signal. The receiving terminal acknowledges receipt of the record by sending a receipt-of-record signal. The terminal sending the message then sends the next record. This process continues until the sending terminal sends an end-of-message signal, which is acknowledged by the receiving terminal. If no more messages are to be sent, the terminals go back to sending IDLE's.

Also included for some STR terminals, is the binary synchronous communication feature (BSC) which provides additional flexibility. Most STR terminals use a special code (that is, four of the eight bits of the character must be on). BSC, on the other hand, can use EBCDIC, USASCII-8, and Six-bit Transcode. BSC also eliminates most restrictions needed for control characters. All data is sent as a serial stream of binary data, which is used to encode the characters making up each transmission code set. The BSC feature is designed for the IBM 2780 Data Transmission Terminal.

The STR type terminal has a much faster and more efficent method of sending and receiving messages than the start/stop type terminals. In most cases, the STR transmission rate exceeds 20 times the start/stop transmission rate.

Data Buffering

All data processing procedures involve input, processing, and output. Each phase takes a specific amount of time. The usefulness of a computer is often directly related to the speed at which it can complete a given procedure. Any operation that does not use the central processing unit to full capacity prevents the entire system from operating at maximum efficiency. Ideally, the configuration and speed of the various input/output devices should be so arranged that the CPU is always kept busy with useful work.

The efficiency of any system can be increased to the degree in which input, output, and internal data-handling operations can be overlapped or allowed to occur simultaneously.

Input is divided into specific units or logical associations of data that enter storage under control of the program. A number of output results may be developed from a single input, or, conversely, several inputs can be combined to form one output result. Figure 101A shows the basic time relationship between input, processing, and output with no overlap of operations. In this type of data flow, processing is suspended during reading or writing operations. Inefficiency is obvious, because much of the available time of the central processing unit is wasted.

Figure 101B shows a possible time relation between input/output and computing when a buffered system is used. Data is first collected in an external unit called a buffer. When summoned by the program, the contents of the buffer are transferred to the main storage unit. The transfer takes only a fraction of the time that would be required to read the data directly from an input device. Also, while data is being assembled in the buffer, internal manipulation or computing can occur in the computer. Likewise, processed data from main storage can be placed in the buffer at high speed. The output device is then directed to write out the contents of the buffer. While writing occurs, the central processing unit is free to continue with other work.

If several buffered devices are connected to the system, reading, writing, and computing can occur simultaneously (Figure 101C).

Further development of the buffering concept has led to the use of main storage as the primary buffer. Data is collected from, or sent to, the input/output devices in words or in fixed groups of characters. Transmission of words is interspersed automatically with computation, but the time required for the transmission of single words is relatively insignificant. The effect is that of overlapping internal processing with both reading and writing. The principal advantage here is that the size or length of the data handled is restricted only by the practical limits of main storage. When external buffers are used, the amount of data handled at any time is limited to the capacity of the buffer. Overlapping operations up to this point have demonstrated a principle of synchronous operation; that is, the action of the input/output devices is made to occur at fixed points in the program and in a sequence established by the programmer.

In some computers, design features allow for automatic interruption of processing by the input/output

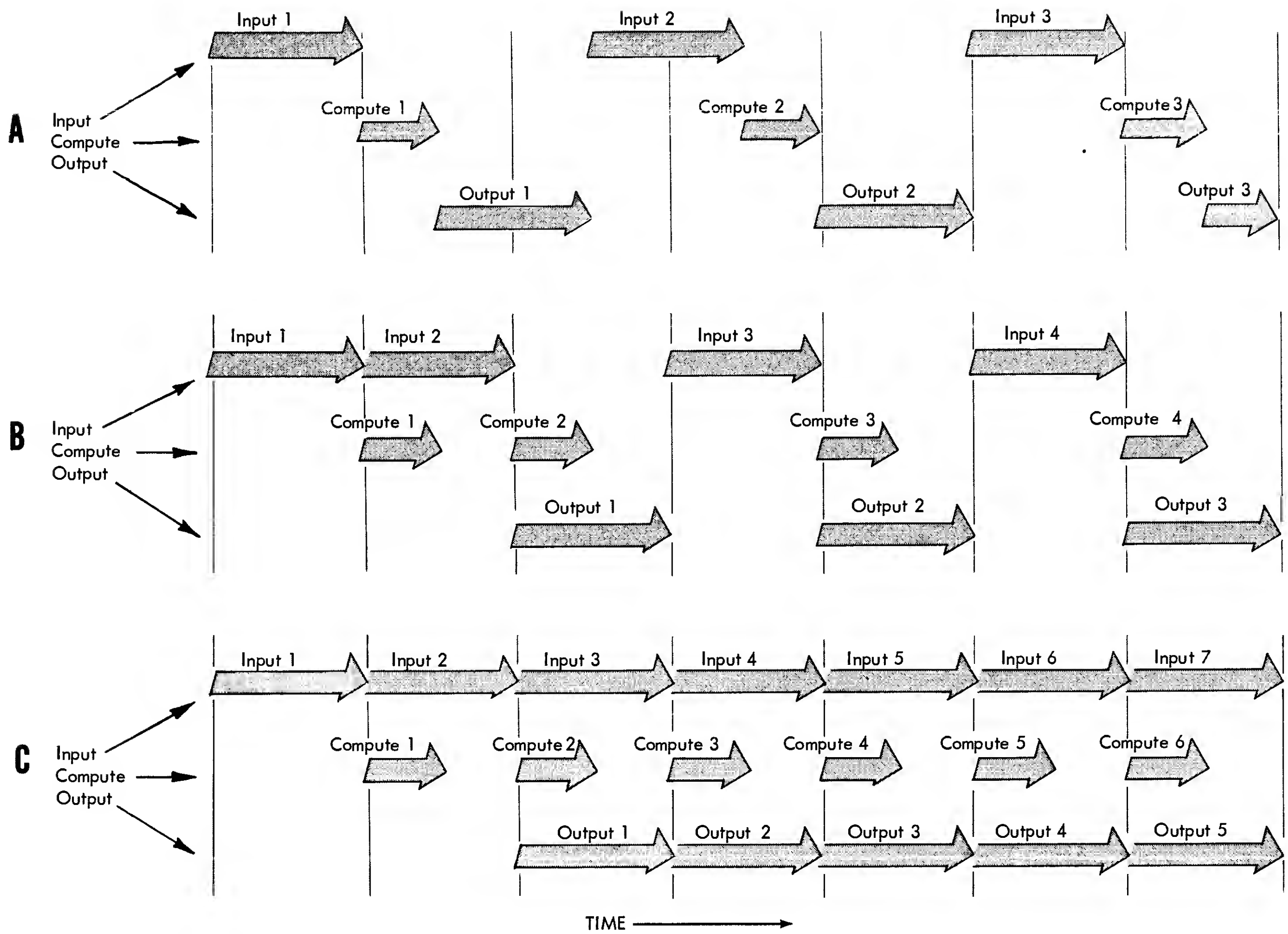


Figure 101. Data buffering

devices; synchronous operation is not required. The input or output device signals the central processing unit when it is ready to read or to write. The central processing unit responds to these signals and either accepts the data as input or transmits the required information as output. In real-time teleprocessing systems, this type of input/output is likely to be nonsequential and unpredictable.

The problem arises of how to fill in the gaps in central processing unit time. The answer is to somehow queue the various tasks and programs to step in and, without interfering with one another, to use the otherwise idle time. This is the basis for multiprogramming —a subject described in more detail under "Programming Systems".

Auxiliary Operation

Input/output and data conversion operations of the data processing system are relatively slow compared with the speed of the central processing unit. Auxiliary, or offline, operation provides a method by which many operations can be performed by machines not directly connected to the system. The advantage is to free the computer of routine, time consuming procedures, thereby providing more time for the prime functions of computing and data manipulation within the central processing unit.

The principal auxiliary operations are those of converting data from cards to magnetic tape, magnetic tape to cards, and magnetic tape to printed reports. For example, all output data from a system could be placed on magnetic tape, the fastest method of recording data from a system. The tape could then, in an auxiliary operation, be converted to cards or printed as reports as the computer continues processing new data.

The importance of auxiliary operation has progressed to a point where it is now usual, with large computers, to use a small data processing system to perform the auxiliary operations.

After data is transcribed to an input medium, the computer system can take over the complete processing and the preparation of results. However, the procedural steps that are to take place within the computer system must be defined precisely in terms of operations that the system can perform. Each step must be written as an instruction to the computer.

A series of instructions pertaining to an entire procedure is called a program. In modern data processing systems, the program is stored internally, and the system has access to the instructions at electronic speeds. Such programs are called stored programs.

Instructions

The computer is directed to perform each of its operations by an instruction – a unit of specific information located in main storage. This information is interpreted by the central processing unit as an operation to be performed.

If data is involved, the instruction directs the computer to the data. If some device is to be controlled – a magnetic tape unit, for example – the instruction specifies the device and the required operations.

Instructions may change the condition of an indicator; they may shift data from one location in storage to another; they may cause a tape unit to rewind; or they may change the contents of a counter. Some instructions arbitrarily, or as a result of some machine or data indication, can specify the storage location of the next instruction. In this way, it is possible to alter the sequence in which any instruction or block of instructions is followed.

An instruction (Figure 102) usually consists of at least two parts:

Operation	Operand
Select	Tape Unit 200
Read	One Record into Storage Positions 1000-1050
Clear & Add	Quantity in Storage Location 1004 in Accumulator
Subtract	Quantity in Storage Location 1005 from Contents of Accumulator
Store	Result in Storage Location 1051
Branch	To Instruction in Storage Location 5004

Figure 102. Instructions

1. An operation part that designates read, write, add, subtract, compare, move data, and so on.
2. An operand that designates the address of the information or device that is needed for the specified operation.

During an instruction cycle, an instruction is selected from storage and analyzed by the central processing unit. The operation part indicates the operation to be performed. This information is coded to have a special meaning for the computer. For example, in a System/360, the letter A is interpreted as "add", the letter C as "compare", SIO as "start input/output", and TR as "translate". Other computers use different coding and numbers of characters or positions to define an operation.

The operand further defines or augments the function of the operation. For example, to perform arithmetic, the storage location of one of the factors involved is indicated. For input or output devices, the unit to be used is specified. For reading or writing, the area of storage for input or output records is indicated or fixed by machine design.

Because all instructions use the same storage media as data, they must be represented in the same form of coding. In some types of computers, such as the IBM 7090, instructions are fixed in length (one word long). In others such as the IBM 1401 and 1410, they may be a variable number of characters long. In System/360, instructions may be any of three lengths: half-word (two bytes), whole word (four bytes), and word-and-a-half (six bytes), depending upon specific instruction operand requirements.

In general, no particular areas of storage are reserved for the instructions only. In most instances, they are grouped together and placed, in ascending sequential locations, in the normal order in which they are to be executed by the computer. However, the order of execution may be varied by special instruction, by recognition of a predetermined condition of data or devices within the system, by unpredictable interruptions from outside the system (teleprocessing input), by hardware conditions that require servicing from a special set of programs, or by other programs that require unusual priority.

The normal sequence of computer operation in a complete program is as follows: The computer locates the first instruction either by looking in a predetermined location of storage assigned for this purpose or

by manual reset. This first instruction is executed. The computer then locates the next instruction and executes it. This process continues automatically, instruction by instruction, until the program is completed or until the computer is instructed to stop.

Two-Address Instructions

In some computers, such as the System/360, instructions have two address portions. Depending on the function of the instruction, the two addresses can, for example, indicate a device to be used and the data to be operated on, or two factors of data to be processed. An output unit to be used could be indicated by one address, and the storage location from which information is to be written could be indicated by the other address. In arithmetic operations, the two addresses could specify two related factors of data, such as multiplier and multiplicand, divisor and dividend, or addend and augend.

Fewer double-address instructions than single-address instructions are required to perform a procedure. This simplifies programming procedures and results in a saving of space in computer storage.

Instructions and Data

The only distinction between instructions and data in main storage lies in the time they are brought into the central processing unit. If information is brought in during an instruction cycle, it is interpreted as an instruction; if brought in during any other type of cycle, it is considered to be data.

The computer can operate upon its own instructions, if those instructions are supplied as data. The computer can also be programmed to alter its own instructions according to conditions encountered during the handling of a procedure. It is this ability to process instructions that provides the almost unlimited flexibility and the so-called logical ability of the stored program system.

Developing a Program

To develop a program, the programmer must know (1) the number of different operations (and their functions) available in the system with which he has to work; (2) the procedure itself, which must be translated, step by step, into computer instructions; (3) the requirements to be met by the result of processing.

The first step in program preparation is a complete analysis of the application to be programmed, including existing and proposed procedures. This analysis is normally accomplished by developing flowcharts and block diagrams, because most data processing applications involve a large number of alternatives, choices, and exceptions.

It is difficult to state these possibilities verbally. Thus, the systems analyst finds use for many types of pictorial representations, including form layouts, control panel diagrams, manpower planning charts, and so on. The two representations to be discussed here are the system flowchart and the program flowchart.

The outstanding value of a flowchart is that it shows a lot at a glance. It graphically represents organized procedures and data flow so that broad essentials and many details, along with their relationships, are readily apparent. Such sequences and interrelationships are hard to extract from detailed paragraphs of text – and, for the program, next to impossible to determine without supporting documentation. In flowcharting, symbols and words support each other; identifications and descriptions, which may be obscure in text, take on more significance when placed in diagrammed sequence. The communication is further improved by consistent use of meaningful symbols (Figure 103) and reasonably uniform techniques.

The template provides basically for two kinds of flowcharts – system and program. This distinction is not new. A system flowchart shows the flow of data through all parts of a data processing system. A program flowchart shows what takes place within a particular program in a data processing system.

System Flowchart

A system flowchart represents an application in which data provided by source media is converted to final media. The emphasis is on the media involved and the work stations through which they pass. In a program flowchart, on the other hand, emphasis is on computer decisions and processing; the chart provides a picture of the problem solution, the program logic used for coding, and the processing sequences.

Because a fundamental program flowchart evolves from a system flowchart, the former inherently has more detail than the latter. Beyond that, a program flowchart is quite explicit, frequently "exploding" into a series of subsidiary flowcharts to an extremely high level of detail.

System flowcharts are likely to be simpler, less "formalized", and easier to draw than program flowcharts; they are also more flexible. A greater choice of symbols is available, as well as more latitude in their use. Examples of the symbols, conventions, and techniques used in a typical system flowchart are shown, with comments, in Figure 104.

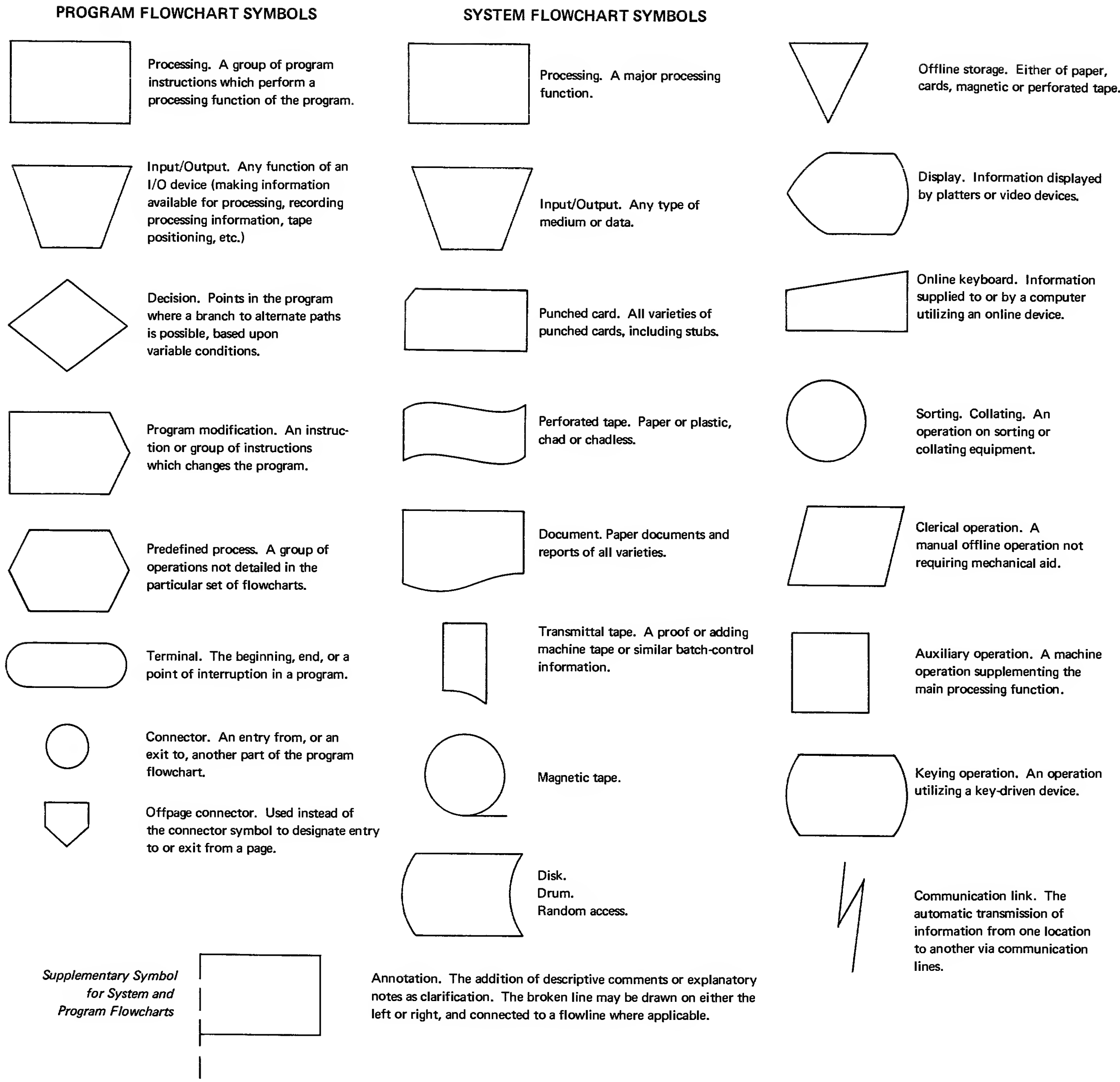
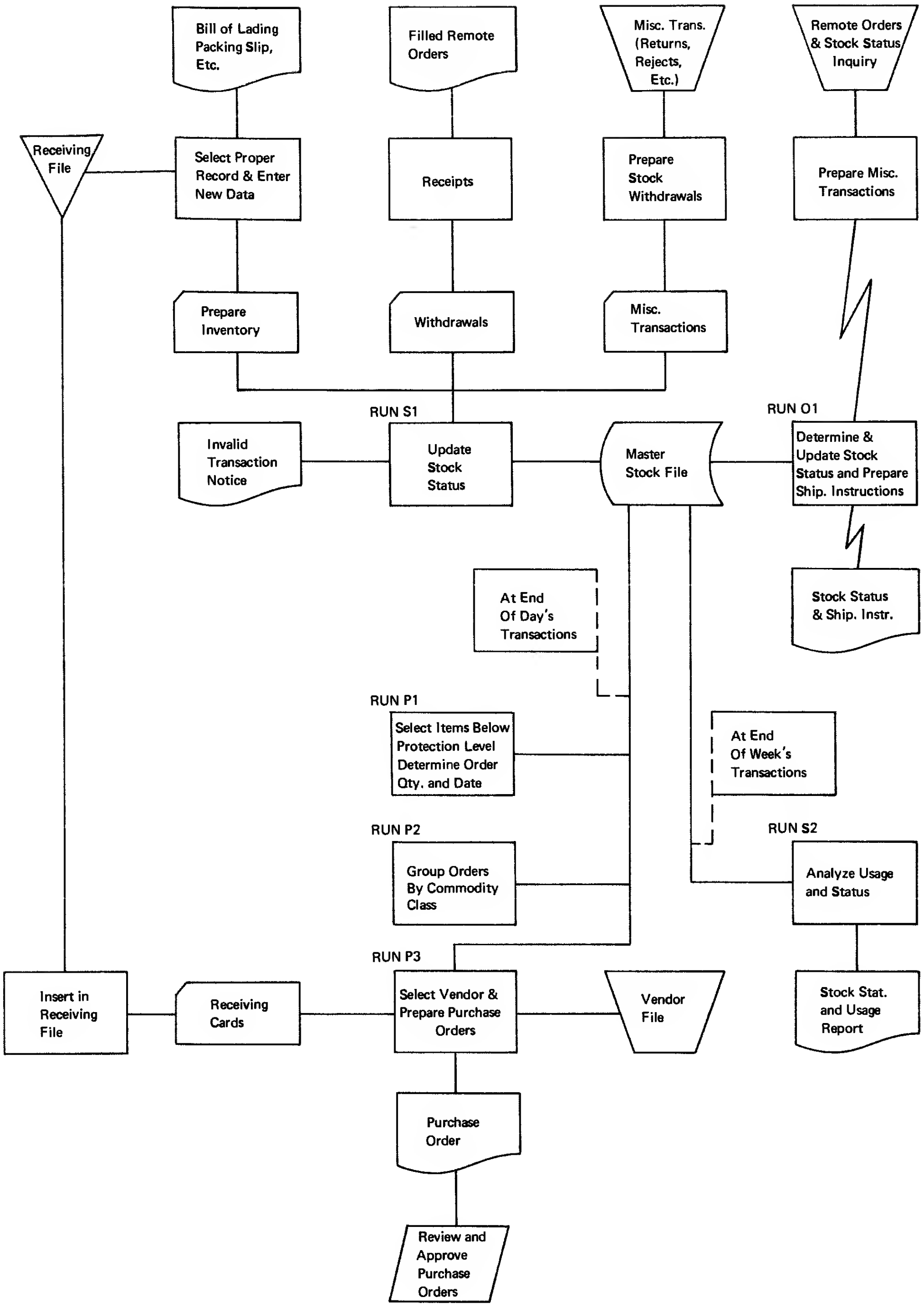


Figure 103. Program and system flowchart symbols



A typical system-flowchart description of an inventory-control application, this chart uses specific symbols for certain processing functions and input/output. The application involves a multiple-warehouse system: items are stocked in a central warehouse for distribution to remote warehouses; all customer orders are received by remotely-located warehouses and transmitted by teletype (communication-link symbol) to the central data-processing installation. The system provides four major groups of operations:

(1) updating stock status [run S1], based on actual transactions; (2) response to inquiries [run O1] from auxiliary warehouses and central warehouse; (3) reorder analysis [runs P1, P2, P3], including purchase-order preparation; (4) weekly analysis reports [run S2] to show slow-moving items, major changes in usage rates, behind-schedule deliveries, economic lot sizes, etc.

Figure 104. Flowchart for an inventory control system

Program Flowchart

Certain aspects of program flowcharts deserve emphasis by themselves, because they are so specialized and so thoroughly integrated into both the routine and the creative phases of programming. To the programmer, a program flowchart is a kind of all-purpose tool. It is the "blueprint" of a program. In program development, the programmer uses flowcharting in and through every part of his task: to marshal and organize his facts on paper; to outline problems, logic, and solutions; to deal systematically with the problem as a whole. He uses flowcharting to build, step by step, his own reference documentation and reminders.

In the development stage of a program, a flowchart serves as a means of experimenting with various approaches – laying out the logic. The programmer starts with symbols representing major functions of a proposed program. He develops the overall logic by combining blocks to depict input/output functions, steps for identification and selection of records, and decision functions.

Once the programmer has tentatively established mainline logic, he usually extracts large segments and describes them in more detail on subsidiary charts. This is like drawing a set of increasingly detailed maps – starting with a general, all-inclusive map, then exploding sections of it on succeeding maps, each map showing greater detail. The technique is called modular program flowcharting. For thorough documentation on this basis, a typical file-maintenance routine could possibly require as many as 80 flowcharts.

When the programmer is satisfied that the procedure is sound, he uses the flowchart as his guide for coding. Sometimes, at this stage, program logic may have to be modified to agree with machine logic, and a chart may have to be redrawn and reverified. Modifications are even more likely during testing, installation, and future operation.

Flowcharting Worksheet

Because most program flowcharts are so detailed, it is a great advantage that they be drawn in a consistent, well-organized format. Such "formalization" is best done on a regular form designed for that purpose. The IBM flowcharting worksheet form (X20-8021) is shown in Figure 105 with a typical chart superimposed on it.

The 11 x 16½" worksheet can be used for all kinds of flowcharts, but it is particularly useful in program flowcharting.

Essentially, what the worksheet provides is an arrangement of 50 blocks with alphabetic and numeric coordinates. Ten horizontal rows are lettered from the top (A) to the bottom (K). Five vertical rows are numbered from left to right – 1 through 5. These coordinates prove helpful in documenting and cross-referencing.

The worksheet has guidelines for the 50 positions, and crosslines indicating the horizontal and vertical centers of each position. These are simply aids for squaring up flowlines, centering symbols in each position, and maintaining uniform spacing between symbols. The result is a neatly arranged, compact chart, but not too crowded. The worksheet itself is printed in light-blue ink so that its guidelines will disappear during photographic copying.

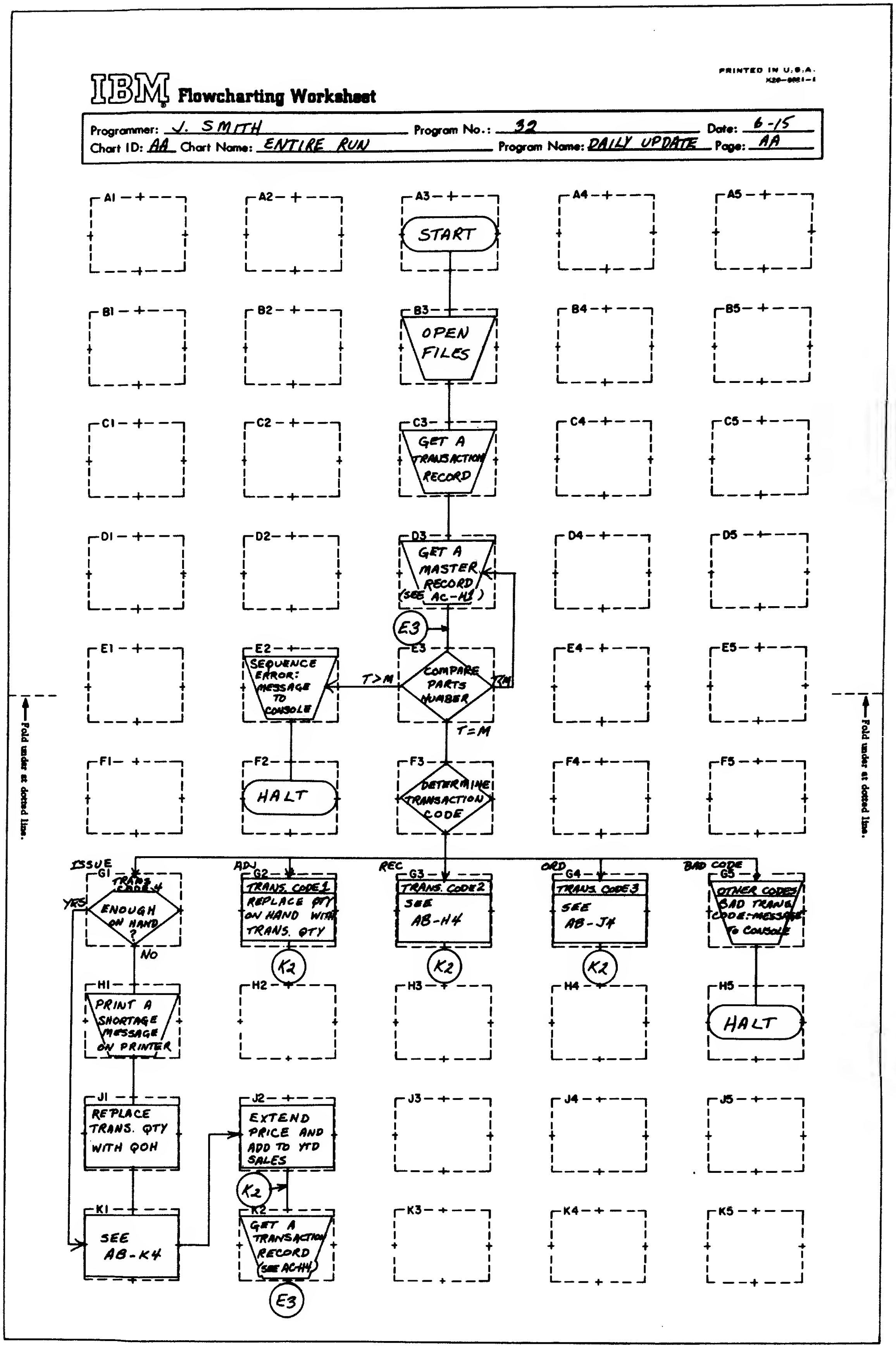


Figure 105. First sheet of a sample program flowchart for an updating run of inventory records.

Miscellaneous Techniques

A few of the more detailed techniques in the drawing of program flowcharts are described briefly and illustrated:

Cross-referencing relates the program flowchart to source language programs (described later). One way is to locate an instruction either by its label or by the page and line number of the coding sheet on which it appears. The cross-reference can be placed above the upper left corner of a symbol, as shown in Figure 106.

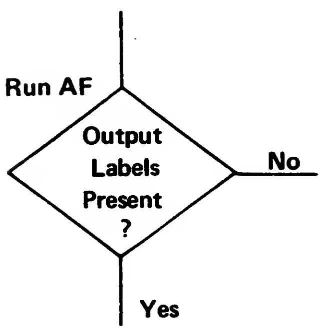


Figure 106. Example of cross-referencing

Striping on a symbol (the horizontal line within and across the upper part of a symbol) indicates a complex logical program unit. Generally, it is used to show that a more detailed flowchart of the program unit exists. (On the template, recommended stripe placements are indicated by heavier grid lines on both sides of the upper part of certain symbols.) Identification of the program unit may be placed above the stripe; a brief description of its function may be placed below the stripe. An example appears in Figure 107.

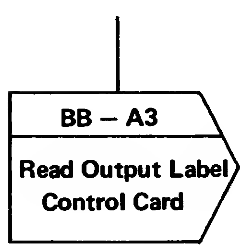


Figure 107. Example of striping

Decision Techniques may be shown in several ways. See examples in Figure 108. These decisions will determine which action is to be performed next by the program (see branch operation).

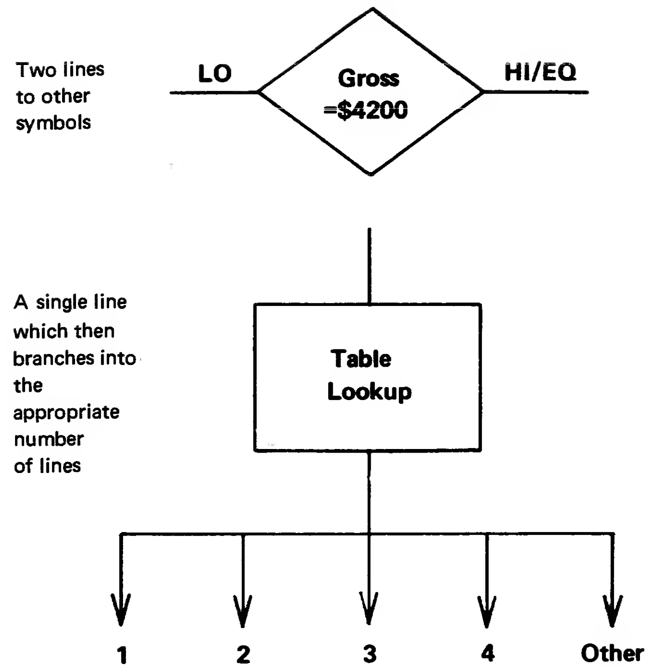


Figure 108. Examples of decision techniques

An example of a typical program flowchart is shown in Figure 105 (superimposed on the flowcharting worksheet form referred to earlier). The program flowchart, for a daily updating run of master inventory records, is based on the system flowchart. The system flowchart (Figure 109) indicates that the updating is from adjustments, receipts, orders, and issues for the day; in addition, a shortage-and-reorder listing is prepared. The system requires no additions or deletions involving the master file.

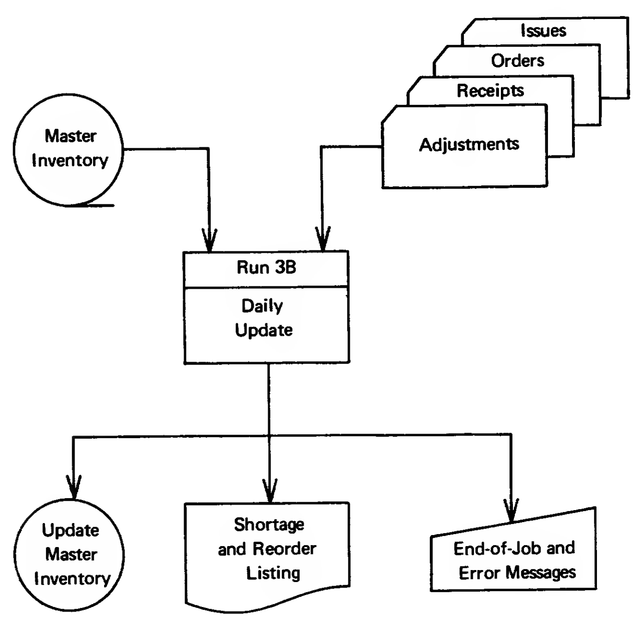


Figure 109. System Flowchart

Flowcharting by Computer

All of the foregoing has dealt with manual flowcharting – drawing the charts by hand. Much progress has been made, however, on the running of flowcharts on a computer. An IBM flowchart program for the IBM System/360 enables the computer to print out, from detailed instructions, a complete flowchart – including machine versions of the same symbols usually drawn by hand. This program is called S/360 Flowchart.

Mechanized flowcharting is particularly helpful in programming, where modifications of a program often require time-consuming redrawing of charts. With S/360 Flowchart, flowcharts manually drawn in a prescribed manner (and coded) can be produced automatically; once produced, they can be modified and rerun with a minimum of time and effort. The flowcharting worksheet in Figure 105 ties in nicely with this implementation: the 50-position grid provided by the worksheet lends itself readily to mechanization.

Reading Data

All data entering the computer system must first be read by an input device and routed to main storage. Each input device is assigned a number to serve as its address in the same way that each storage position is also assigned a location address.

A data processing procedure is normally concerned with entire files (called "data sets" in Operating System/360) of records on one or more of the input media. These files are either fed to the input device, where the computer has access to them, or read directly from a secondary storage unit. To read a record from a file, one or more instructions in a program activate the input device and place the record in main storage.

At this point, it must be determined exactly where in storage the incoming record is to be placed, and an instruction must direct the machine to send it to this predetermined location. Also, in the plan of manipulation, it is necessary to know at all times where to find information as needed in the successive stages of processing.

These considerations involve the allocation of storage space for specific purposes in a logical and convenient manner. For example, particular fields or quantities may be used for computation. The instructions to be used later must specify the location in storage where this information from each record can be found.

When a data processing system includes an operating system of programs to take care of placement of incoming records and fetching of stored records as needed, the problem programmer is not concerned with the location details.

The reading operation performs the following distinct functions:

1. The input device is selected and made ready before actual reading begins. The device chosen is the one that has access to the proper file records as determined by the programmer. This device is selected by specifying its assigned code number or address.
2. The read instruction causes the previously selected unit to carry out the transfer of a record to the storage of the computer. The record is placed in a particular storage area reserved for this purpose and is then available for further processing. A number of input areas may be assigned to handle several related records at once (for example, a master record and its corresponding transaction detail).
3. The order of read instructions in the program determines the sequence in which the files are read. Other instructions later compare records from separate files to determine the relationship of detail to master, detail to detail, and so on.
4. The number of records to be placed in storage at one time depends on the construction of the files, the type and length of records being handled, and the available storage capacity.

Calculating

Once data has been read into the computer system and placed in known locations of storage, calculation can begin. Each computer is capable of performing addition, subtraction, multiplication, and division, either as built-in operations or under program control. For most commercial applications, these operations are adequate. Even in many of the more advanced scientific procedures, the most complex equations can be reduced to steps of elementary arithmetic. However, many specialized operations can be performed by some systems to make the solving of mathematical problems easier.

In every operation of simple arithmetic, at least two factors are involved: multiplier and multiplicand, divisor and dividend, and so on. These factors are operated on by the arithmetic unit of the machine to produce a result, such as a product or a quotient. In every calculation, therefore, at least two storage locations are needed. One quantity is usually in main storage, the other in a register. In System/360, both quantities may be in registers.

A calculation can be started by placing one factor in the register and at the same time clearing this unit of

any previous factors or results that may be contained there. The address part of the instruction specifies the storage location of the first factor; the register is implied by the operation. In some computers, more than one register is available for calculation. In this case, the address must also specify the register to be used.

When one of the factors is properly placed in the accumulator or other suitable register, the actual calculation is executed by an instruction whose operation part specifies the arithmetic to be performed and whose operand is the location of the second factor. The computer acts upon two factors, one in the register and the other in storage, and produces a result in either place, as directed.

The result may be moved to a storage area, as a field in some record. A field is a related arrangement of characters or digits to represent a quantity, amount, name, identity, and so on.

Any practical number of calculations can take place on many factors in a single series of instructions. That is, a factor may be placed in a register and multiplied, and several other factors may be added to or subtracted from the product. Division can then be executed, and other operations of adding and subtracting can proceed using this quotient. Intermediate results can be stored at any time.

For example, a field containing employee hours worked can be placed in a register and multiplied by hourly rate to produce earnings. Piece work and bonus amounts may then be added to develop a total regular earnings amount, which is stored in the pay record. Total regular earnings may then be divided by hours to produce an average hourly rate. This rate is multiplied by 1.5 overtime hours to produce overtime earnings. Total gross pay is then calculated and stored. Taxes are computed using the calculated gross pay; other payroll data is accumulated using the amounts as they are calculated. Intermediate results of tax amounts and deductions, and, finally, net pay are all stored in the pay record.

Operations of shifting and rounding the contents of the register are also provided to adjust, lengthen, or shorten results. With these operations, decimal values may be handled and directions for placing of the decimal point may be given to the computer.

All calculations must take into account the algebraic sign of factors in storage or associated registers. Consequently, the computer system is equipped with some provision to store and recognize the sign of a factor.

If records are made up of fixed words of data, one position of the word is designated as a sign position and automatically accompanies the word. Registers also include either a special sign position of storage or a sign indicator that is available to the programmer. In this way, the sign of results can be determined, together with the effect, after calculations. The computer follows the rules of algebra in all basic arithmetic calculations.

The size of words, quantities, and values depends upon the design of each particular system. The exact rules governing the placement of factors, size of results, etc., vary somewhat from system to system. In all cases where a result is expected to exceed the capacity of the register, the programmer must arrange his data to produce partial results and then combine these for totals. Other operations of scaling may be executed so that very large or small values and fractions may be handled conveniently. Computers designed primarily for mathematical applications generally include a series of specialized arithmetic operations for this purpose. (See "Floating-Point Operation".)

Calculation is carried out in all computer systems at much higher rates of speed than input or output, because reading and writing require the use of mechanical devices and the movement of documents, while calculation is performed electronically. In many commercial applications, calculation is relatively simple, and the overall interleaving of the system is usually governed by the speed of the input/output units. In mathematical applications, the situation is reversed; calculation is complex and involved, and high calculating speeds are essential. The design of any particular system must achieve a realistic balance between calculating and record-handling ability.

Logical Operations

The sequence in which a stored program computer follows its instructions is determined in one of two ways: either it finds the instructions in consecutive storage locations, or the instruction operand also designates the location of each following instruction. If instructions could be followed sequentially only in a fixed pattern, a program would follow only a single path of operation without any possibility of dealing with exceptions to the procedure and without any ability to choose alternatives on the basis of special conditions encountered in processing data. Further, without some way of resetting the computer to repeat a given series of instructions, it would be necessary to have a complete program for each record in a file.

Consider the program illustrated by the block diagram in Figure 110.

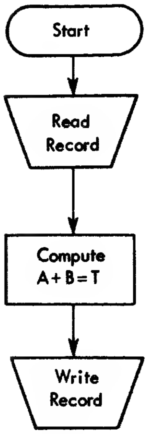


Figure 110. Program flowchart A+B = T

These instructions taken alone compute T for only one record. But, by returning to the first instruction, any number of records can be processed, repeating the same program as a loop. For this purpose, another instruction is given to return to the first instruction (Figure 111). Once this program is initiated, it will continue to run until there are no more records to process. Program loops are common, and they can be terminated in many ways.

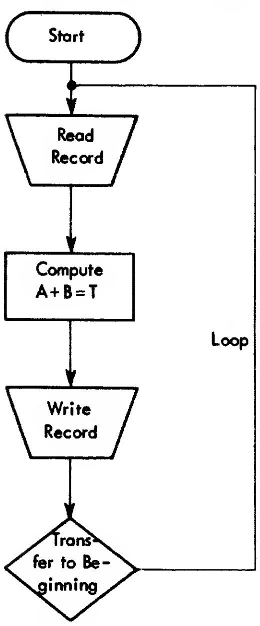


Figure 111. Program loop

For example, the computer may be instructed to examine T each time it is computed and to go to a certain routine when the value of T becomes negative (Figure 112).

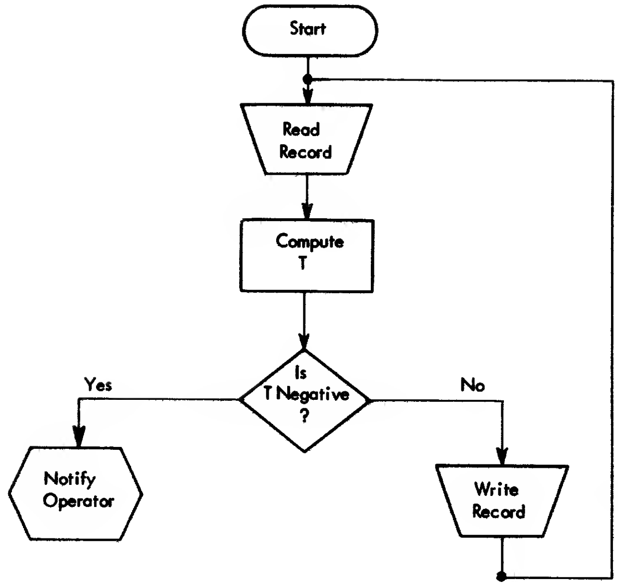


Figure 112. Conditional transfer

In this case, the instruction becomes a conditional transfer. The program is repeated only if some predetermined condition has been satisfied. The computer may also be instructed to execute the program for ten records and then go to a certain routine (Figure 113).

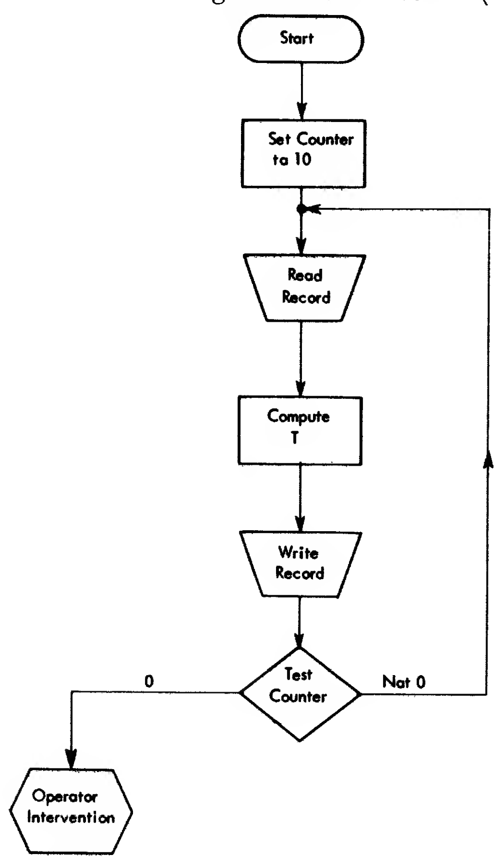


Figure 113. Record count conditional transfer

It is assumed that the constants 10 and 1 are in the computer and that 1 is subtracted from 10 each time the loop is completed. After ten times around, a zero will be found in the location that contained 10. A transfer or branch then terminates the loop.

The conditional transfer or branch operation may be used to cause a special purpose subroutine to be executed outside the normal or straight-line path of the program. This subroutine is executed only when a predetermined exception or condition is noted by the machine.

One common example of the subroutine is checking the accuracy of records as they are read from, or written on, magnetic tape. As each record enters or leaves the central processing unit, a read/write error indicator is examined. If the indicator has been turned on, the computer is instructed to enter a subroutine of instructions that attempt to correct the error. The program logic for such an operation — the reading only — is shown in Figure 114. A similar loop might also be included for writing.

When a reading error is detected, a branch is effected to the error subroutine. A counter is reset to the quantity 10 to count the number of times a reread will be attempted. The tape is backspaced over the error, and a second read instruction is given. Another check is made to determine whether this operation is correct. If it is, a transfer returns to the main program, where computing continues.

If the error persists, 1 is subtracted from the counter, and the counter is tested for 0. The error loop is again entered, and a second reread and check are executed. The machine can reread ten times. If the error is not corrected, the program transfers to another routine, where it goes through some sort of procedure to log the error record and then return to read in and process the next record. With Hypertape and 2400 series nine-track tape, however, the cyclic redundancy character feature corrects single-track errors automatically, thus eliminating the need for this type of program, in many instances. Also, input/output control systems and operating systems (programming systems supplied by IBM) are available for most data processing systems, eliminating the need for the problem program to include this type of checking.

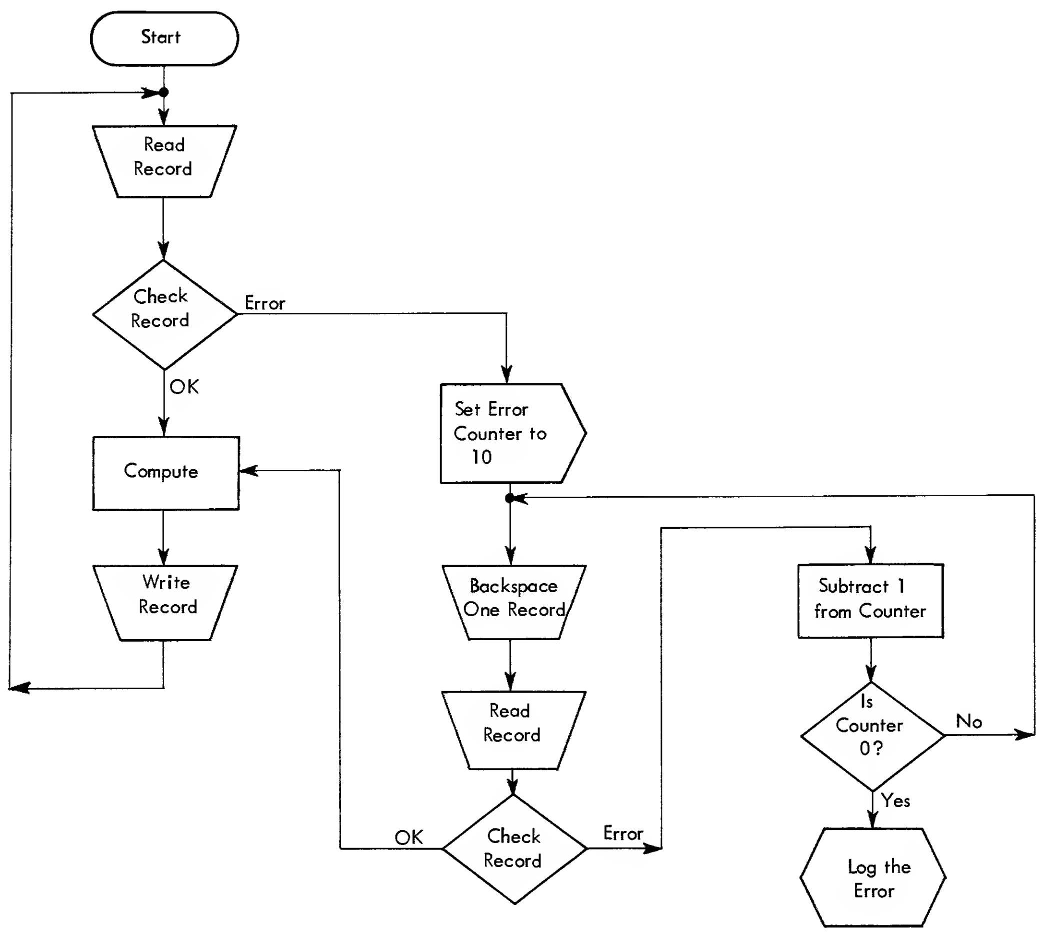


Figure 114. Read error loop

A program can also be arranged so that the machine can recognize one or more types of records as they are processed from a single file. The method of computation can be varied according to the type of record in storage. This procedure is common when a number of types or classes of transactions are processed against a single master file (for example, in an application of file maintenance).

Assume that a file (data set) of master stock status records contains quantities that reflect the number of parts available for manufacturing planning. The records also have considerable other information pertaining to the status of inventory, but for purposes of illustration, this example is concerned only with those fields used to show availability. These fields are:

Quantity in stock
Quantity on order
Quantity available

Transactions that affect the status of the parts availability originate daily. These transactions are punched in cards with an identifying digit code for each type of activity.

Codes are as follows:

Code 1 Receipts
Code 2 Orders
Code 3 Withdrawals
Code 4 Adjustments plus
Code 5 Adjustments minus

As each transaction is placed in storage, it is analyzed by code to determine the class to which it belongs (Figure 115). A branch instruction then transfers

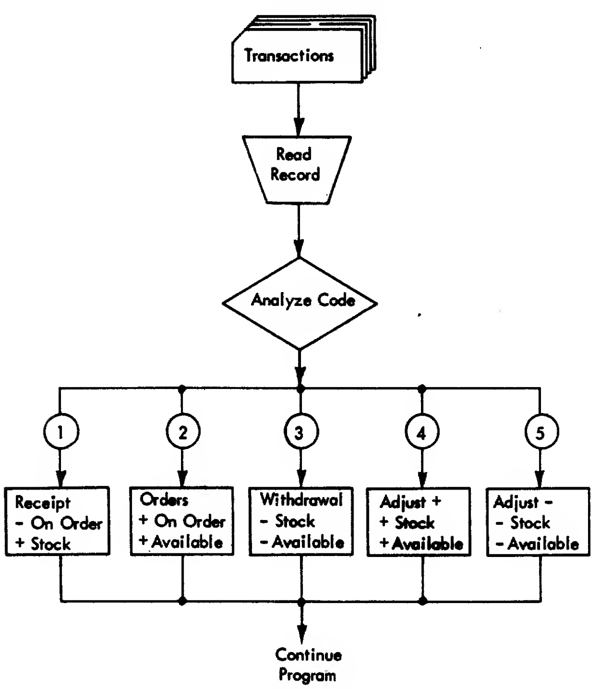


Figure 115. Branching by code

to the proper program subroutine to calculate availability and to adjust the corresponding master record. Reading and writing of the adjusted master record are not shown in the flowchart.

Comparing

The ability of the computer to make limited decisions on the basis of programmed logic is substantially extended by operations of comparing. Such operations enable the computer to determine whether two data fields in storage match, or whether one is lower or higher than the other. The basis of comparison is set according to some predetermined sequence built into the circuitry.

The sequence may be considered to be a normal filing order of records of all types. For example, the familiar ascending sequence of the digits 0-9 assumes that the digit 9 is the highest digit of the series. In the same manner, the letter Z is assumed to be the highest letter of the alphabet. To the computer, therefore, as in any file, the number 162 is higher in sequence than the number 159, and the name Jones is lower than the name Smith. Special characters, such as / @ * , or –, may also be included, because all computer data has a value that can be compared with any other value. This is known as the collating sequence.

Comparing operations are used to program the sequence checking of files, sorting procedures, or the rearrangement of records in some desired order. The comparison of an identifying field in one record with that of another enables the computer to handle a number of associated files in one processing procedure, provided that all files are in sequence by this common field.

One or both fields are placed in a storage register(s). The compare instruction then compares the first field against the second. (The second field, in System/360, may be in a second register; in other systems, it is in a main storage location.) The results of comparison are registered as high, low, or equal, by indicators or triggers that may then be interrogated to determine their condition. If the indicator is on, a branch (transfer) instruction transfers the machine to a subroutine that will continue processing according to the result of the comparison.

Figure 116 shows a typical program arrangement for sequence checking a single file of records. All records in the file are assumed to be in ascending sequence by account number.

An input area is set aside in storage, where records are received, one at a time, from an input unit. A second area is also reserved in storage to store the account number from the preceding record. The purpose of

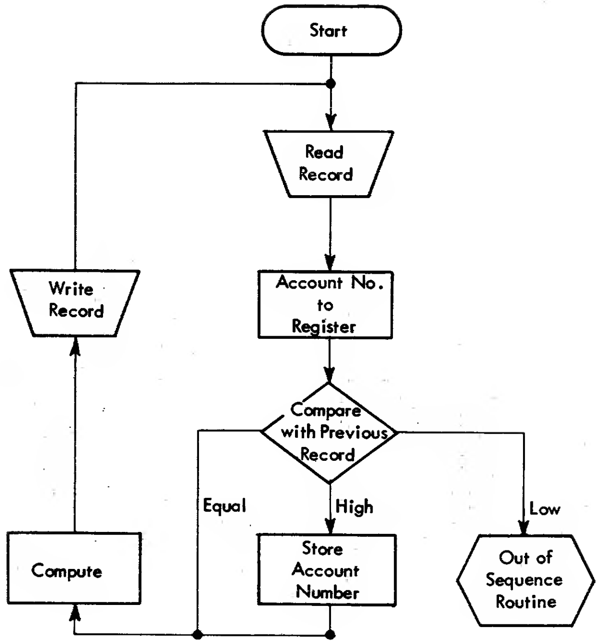


Figure 116. Sequence checking

this area is to allow comparison of the account number of the incoming record with the corresponding field of the previous record.

If the file is in ascending sequence, the incoming record should always be higher than the record that preceded it. When duplicate records are encountered, the incoming record is equal to the preceding one. If any incoming record is lower than the previous record, it is recognized as an out-of-sequence condition, and the program transfers to a subroutine to take corrective action. After each high comparison, the account field is placed in storage, where it may be compared with the next record.

Instruction Modification

Some of the preceding examples have shown how branching or transfer instructions can cause the computer to follow a varied path through the program. The routine to be executed depends on the result of a previous comparison or a test of indicators that have been set by a zero balance, an error condition, and so on.

Another method of varying the program is by changing or modifying the operation part of the instructions themselves. Instruction modification, for example, can be used to set up a program switch that can cause the machine to take one of two alternate paths. The switch is turned on or off by instruction. The use of the switch is shown in Figure 117.

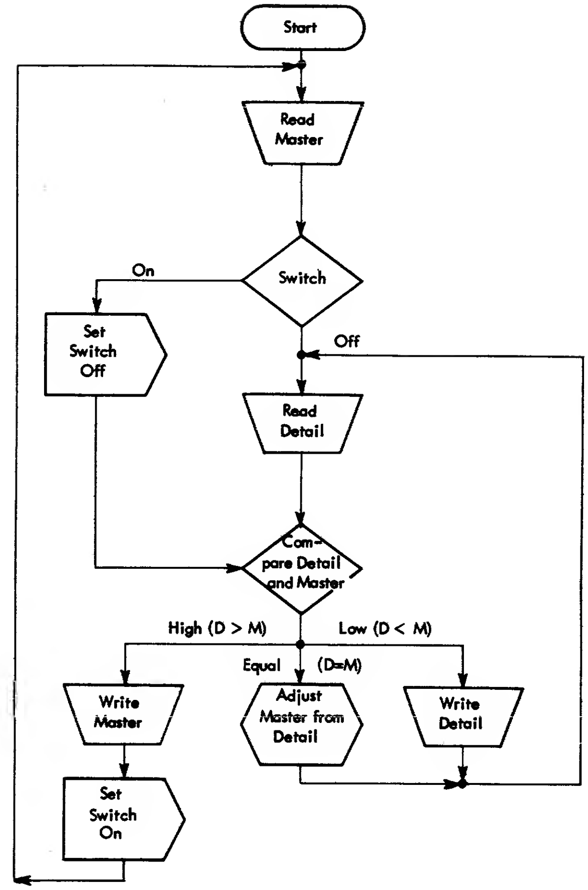


Figure 117. Program switch

Assume that two files (data sets) are being read. They are in sequence by a common identifying field, such as part number, account number, or employee number. One file is a master file; the second is a transaction file that represents adjustments to the master. Three conditions may be encountered in applying the transactions to the corresponding master file (data set):

1. One or more transactions may match a single master record.
2. There may be no transactions for a master record.
3. There may be transactions that do not match a master – these are errors or new additions.

It is necessary to process the two files (data sets) in step; that is, each transaction record must be compared against a corresponding master record, if there is one. If several transactions apply to the same master record, the transaction file (data set) must continue reading without reading a new master record. Conversely, if a master record is read in without a corresponding transaction, this record is written out unchanged, and the following master is read in. The reading and writing of master records continue until a matching transaction is found.

The flowchart in Figure 117 shows that one master record is read in first. A switch instruction is inserted between the reading of the master and the transaction. As operations begin, this switch is turned off, allowing one transaction to be read in. The identifying field of the transaction is compared against the master. If it is equal, the master is adjusted, and a second transaction is read in. If this transaction is not to be applied against the master (which is still in storage), it should be high when compared. The previously adjusted master is then written out, and the switch is turned on. A new master is then placed in storage, but because the switch is on, a transaction is not read; instead, the machine transfers directly to the compare instruction. The switch is turned off each time this happens. Operation continues with comparison for each new record placed in storage. If a transaction is low, it is written out on a separate output unit, and a new transaction is then read in.

The switch, when on, has an operation part specifying an unconditional transfer. The address part is the location of the compare instruction. To turn the switch off, the operation part is changed to no operation. In this case, the machine ignores the instruction and proceeds to the following instruction: read a transaction.

Address Modification

The address portion of instructions may also be treated as data. An instruction address can be modified by arithmetic, it may be compared against other addresses or factors, or relocated in storage at will. Address modification serves two purposes:

1. The total number of instructions in a program may be reduced, conserving storage capacity for data or other factors. One instruction, or a single series of instructions, can serve to address variable locations in storage.
2. A basic flow of work controlled by the program can serve as a pattern of procedure that can change as required by the entry data, the result of calculation, various error conditions, end-of-file detection, and so on.

For example, the address portion of an instruction that gets information from a table may be modified by the value of a character in a register (useful procedure for handling character translation tables) or by the value of a terminal line number (essential in setting up correct conditions for input for a given line).

Indexing

In many computers, the address portion of an instruction can be modified by adding or subtracting variable quantities contained in one or more special purpose counters. The counter may be called an index register when it is set aside specifically for this purpose, or it may be a predetermined location in core storage called an index word. A computer may have several index registers or a number of storage locations for index words. Both the index register and the index word perform identical functions; however, the word is usually more accessible to the program and, consequently, offers more flexibility in its use.

Computers with an indexing feature use an expanded instruction format that allows a particular register or word to be specified as a part of the instruction operand.

Assume that 50 quantities are placed in ascending word positions of storage from locations 1001 to 1050 inclusive and that these quantities are to be added to the contents of a register. Without indexing or address modification, it is necessary to repeat an add instruction 50 times with the address of each instruction incremented by 1, as ADD 1001, ADD 1002, ADD 1003, and so on.

With indexing, the add instruction can be written as ADD 1051, with the address decremented by an index register containing the quantity 50. The address remains 1051, but the computer calculates an effective address of 1051 minus 50, or 1001. When the add instruction is executed, the contents of the index register are also decremented by 1, leaving a remainder of 49. When the same add instruction is reexecuted and is again decremented by the contents of the same index register, the effective address is 1051 minus 49, or 1002. If a program loop is formed to repeat this process, the effective address of the add instruction is stepped up 1 each time it is executed (as the index register is stepped down). When the index register equals 0, all 50 quantities will have been added, and the loop is terminated. The computer has consequently performed 50 operations using the same instructions.

Figure 118 is a flow diagram of the index loop. The first instruction places the quantity 50 in index register 4. An add instruction, with an address 1051, also specifies as part of its operand a designation that the given address is to be modified by the quantity contained in index register 4.

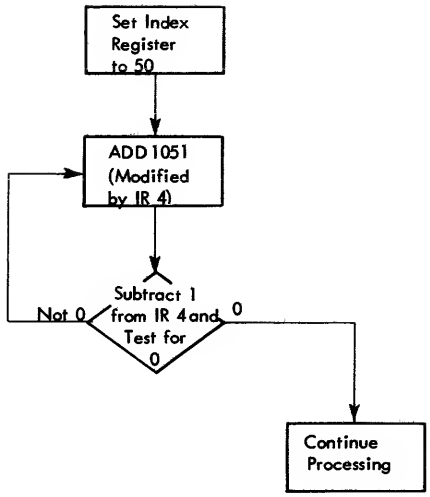


Figure 118. Index loop

The next instruction is branch on index, which means: reduce the contents of the index register by 1; if the contents of the register are greater than zero, branch to repeat the add instruction; if the contents of the index register equal zero, continue to the next instruction in the program.

The indexing feature greatly simplifies programming of repetitious calculations or other operations and reduces the required number of instructions.

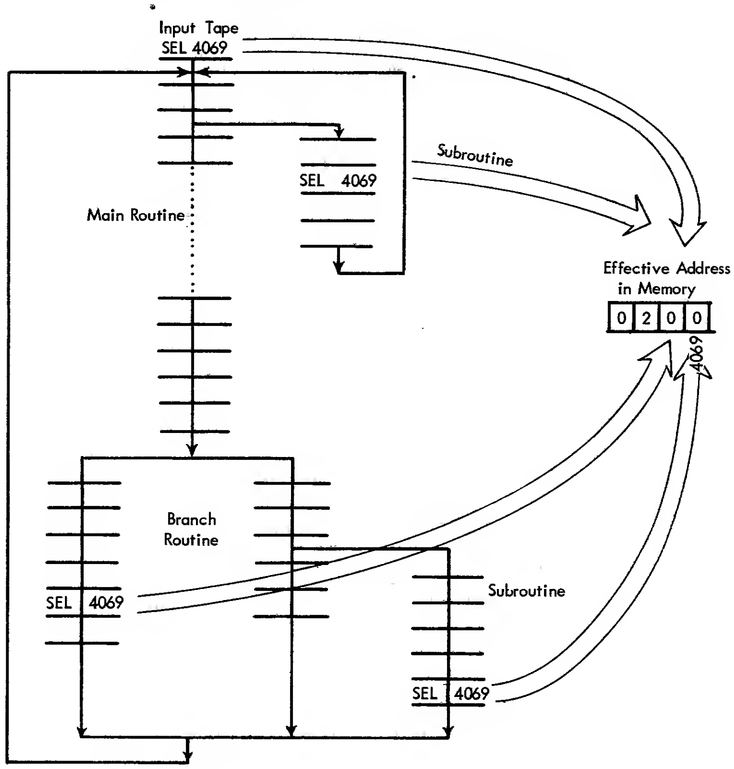


Figure 119. Indirect address

Indirect Addresses

All instruction addresses discussed in preceding illustrations are classified as direct – that is, they refer directly to the location of data or other instructions in storage, they select a machine component, or they specify the type of control to be exercised.

Addresses may also be indirect. Such an address can refer only to a storage location that contains another address. The second address, in turn, refers to the location of data, a machine component, or a control function.

Indirect addressing is particularly useful in performing address modification. For example, in a program it may be necessary to refer a number of instructions to a value that changes during the program run. Without indirect addressing, a number of modification instructions would be needed.

However, if the instructions are indirectly addressed to one core storage location, that location can contain a single address: the address of the values being used by the program. Therefore, to change or to modify all instruction addresses, it is necessary only to modify the single effective address to which the instructions refer (Figure 119). Any number of indirect addresses throughout a program may refer to a single effective address. In Figure 119, each indirectly addressed instruction (SEL 4069) would bring in the contents of core location 200 instead of location 4069.

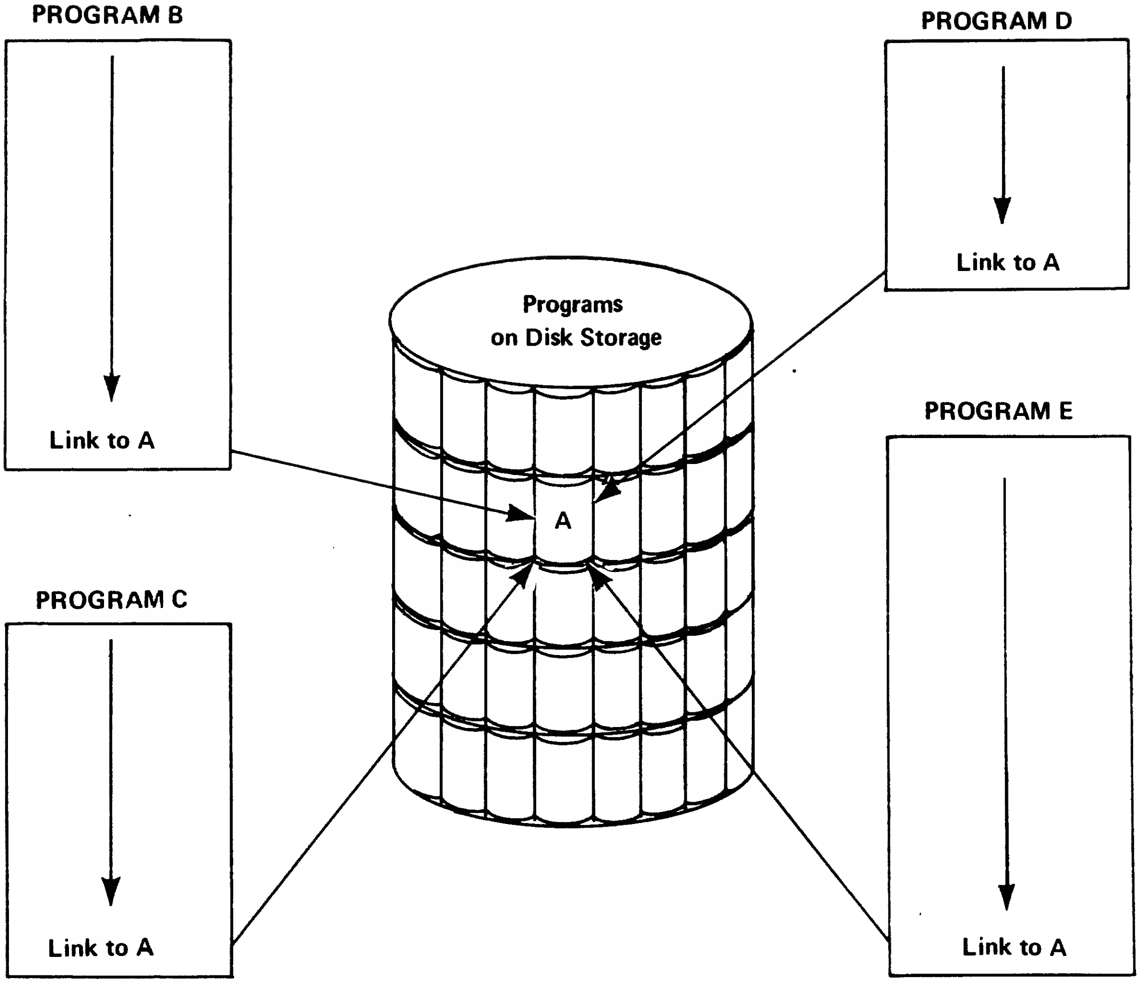


Figure 120. Modular programs with linking

Linking

Up to now, we have discussed conditional branches (transfers) and, most recently, have introduced the principle of indirect addressing. In more modern computers, it is usually possible to "link", with one instruction, to a subroutine so that the program can come back to its point of departure from the main program after finishing with the subroutine, and not have to store, unload, reload, and perform other housekeeping jobs. By using indirect addressing in the linking procedure, a programmer can independently write many subroutines; the link instruction then causes the computer to insert the desired effective return address in the appropriate instruction of the subroutine each time the subroutine is entered. We might imagine each of these subroutines as a data set (comparable to a book in a library with disk storage acting as a revolving bookcase, Figure 120). The exact linking method differs from computer to computer: a simplified procedure is outlined below.

Suppose that many different types of reports are to be printed at a remote IBM 1050 terminal and that the computer is also polling other remote terminals for input. Assume also that as soon as the internal processing of the information for each of these types of output reports is completed, the information is stored on disk files in standard BCD format. Each time a message is taken from a disk file, it must be translated into a slightly different 1050 BCD code format. The program sequence might be:

INSTRUCTION NUMBER		
1000	Link to:	Read message 1 from disk
1001	Link to:	1050 translation routine
1002	Link to:	Process input from line 1, if any
1003	Link to:	Write message 1 to 1050 on line 2
1004	Link to:	Process input from line 3, if any
1005	Link to:	Read message 2 from disk
1006	Link to:	1050 translation routine
1007	Link to:	Process input from line 4, if any
1008	Link to:	Write message 2 to 1050 on line 2
•		•
•		•
•		•

In this case, the first time the program links to the 1050 translation routine, either the programmer puts the address 1002 as the return address or the computer does it automatically for him. The second time, the return address is 1007. In step 1002, the program links to a routine to test for a certain condition. This means that the subroutine ends with another subroutine. If there is no input from line 1, as the result of some indicator being tested, the program branches to pick up the indirect address (stored previously by either the programmer's program or the computer). If there is input from line 1, the last instruction of the subroutine will have to have the indirect address that points the way back to 1003.

Factors in today's computers that make this type of programming indispensable are internal interrupt systems that permit processing to continue until some type of input/output activity is ready to start, or that permit a second program to be processed while a first one is waiting for an error condition to be rectified or an I/O operation to take place, and external interrupt systems that permit teleprocessing interrupts from remote terminals.

It is easy to see, from the simplified programming example described above, and from Figure 120, that programs that are subject to interrupts must consist of short subroutines with a hierarchy of linking, and, if the computer itself does not insert the return address in the "linked to" subroutine, the programs must build tables of indirect addresses.

Depending upon the design of the computer, more or less of this linking procedure can be done automatically. The programmer must store the subroutines and call for them when the computer does not do it automatically.

Interrupts that can occur as the result of a normal anticipated computer function, such as an interval timer timing out or the completion of an I/O operation, can cause an automatic link to an address that is in a fixed location in main storage. Other types of links that cannot be anticipated by machine design require that the program itself do the linking and maintenance of the indirect addresses.

Chaining

Chaining has several possible connotations in present-day IBM computers. In general purpose computers, such as System/360, chaining refers to I/O channel program-linking of commands or data addresses as "command chaining" or "data chaining". Linking described a system for keeping track of the return addresses at the end of a program subroutine; chaining, on the other hand, is a similar technique of programming for a "subroutine" of input/output channel commands to be carried out, independently of CPU activity. This can take place in a computer system that permits the programmer to include, as part of one command, the address of the next command that the channel is to execute when it finishes its current input/output operation. The next command may be either to start a different input/output operation or to transfer data to or from a different location in main storage.

Another concept of chaining is one that exists in computers designed especially for communication control. Here, chaining is a system of automatic block allocation, whereby incoming data from each line is automatically stored wherever main storage space is available, with the computer automatically inserting the address of each new block assigned to a message in the last two characters of the previously assigned block, wherever it may be. On output messages, chaining is not quite so automatic. The program inserts the address of each new block (of 32 characters) in the last two characters of the previous block. This type of chaining eliminates the need for allocating ahead of time a static amount of storage space for each line, space that may either be insufficient or go to waste because of inactivity. Chaining, in this sense, is an efficient form of data buffering, discussed earlier under "Input/Output Devices".

The capability of computers is expanding at a fantastic rate, and the technology of utilization and control is advancing at an equal pace. These improvements in techniques are as vitally important as the design of the data processing system itself. To a large extent, the future of computers depends not only on increases in speed, logical ability, and storage capacity, but also on the efficient use of these facilities as they are made available.

Programming languages have been developed by IBM to meet both present and future requirements of computer application.

Program Preparation

A computer program represents much more than a set of detailed instructions. It is the outcome of a programmer's applied knowledge of the problem and the operation of the computer system.

Problem definition, analysis, and program flowcharting (see preceding section) are the first steps in program preparation. They are usually carried out independently of the computer and the programming system.

Some or all of the following must be considered to prepare even the simplest program (without the aid of preprogrammed input/output and monitoring support, which will be discussed later):

1. Allocation of storage locations to data, instructions, and related information.
2. Conversion of original data to an input medium.
3. Availability of reference data, such as tables, files, or constant factors.
4. Requirements for accuracy, and methods of checking and auditing.
5. Ability to restart the system in case of unscheduled interruptions or error conditions.
6. Automatic monitoring of the system to ascertain that the required input and output devices are connected and available for operation.
7. Housekeeping procedures that preset indicators, switches, and registers; that type operator messages; and that check file labels.
8. Format of output data with provisions, if required, for later conversion to cards, printed reports, or displayed reports.
9. Availability of program routines that have been used and tested in other procedures and that may be used to advantage in the current procedure.
10. Conversion from the decimal number system to binary and from binary to decimal, plus whatever other conversions are necessary to the internal numbering system of the computer and codes used in input and output.
11. Editing of data with provision to record exceptions that cannot be processed.

Machine Coding

Figure 121 shows the basic relationship between the computer and the programmer when the program is written in actual machine coding. The problem is first analyzed in terms of operations that the computer can perform. The program is then written in machine coding by the programmer, who supplies tables, formulas, codes, or other reference material necessary for the specific application.

The problem then becomes input data, and the computer – by calculation or other operations – produces useful output.

A number of difficulties arise when the program is written in actual machine coding:

1. All instructions must be coded in machine language. With some computers (such as the IBM S/360, which uses binary representation in fixed words), this method of programming becomes especially impractical.
2. Instructions must be written in the exact sequence in which they are to be executed by the computer. If one or more instructions are omitted by error, all succeeding instructions must be relocated in storage to make room for insertions, or some branching technique must be used to incorporate the new instructions. This clerical program accounting for all storage areas must be carried on entirely by the programmer.
3. The full burden of logic and program organization is placed directly on the programmer.
4. Previous experience – tested programs that might be used in part of the procedure – is difficult to work into the new program. Such programs must be linked to the new program by additional handwritten instructions.
5. The programmer must understand the computer in detail. He must know the location of each indicator or register, and he must program their functions entirely.

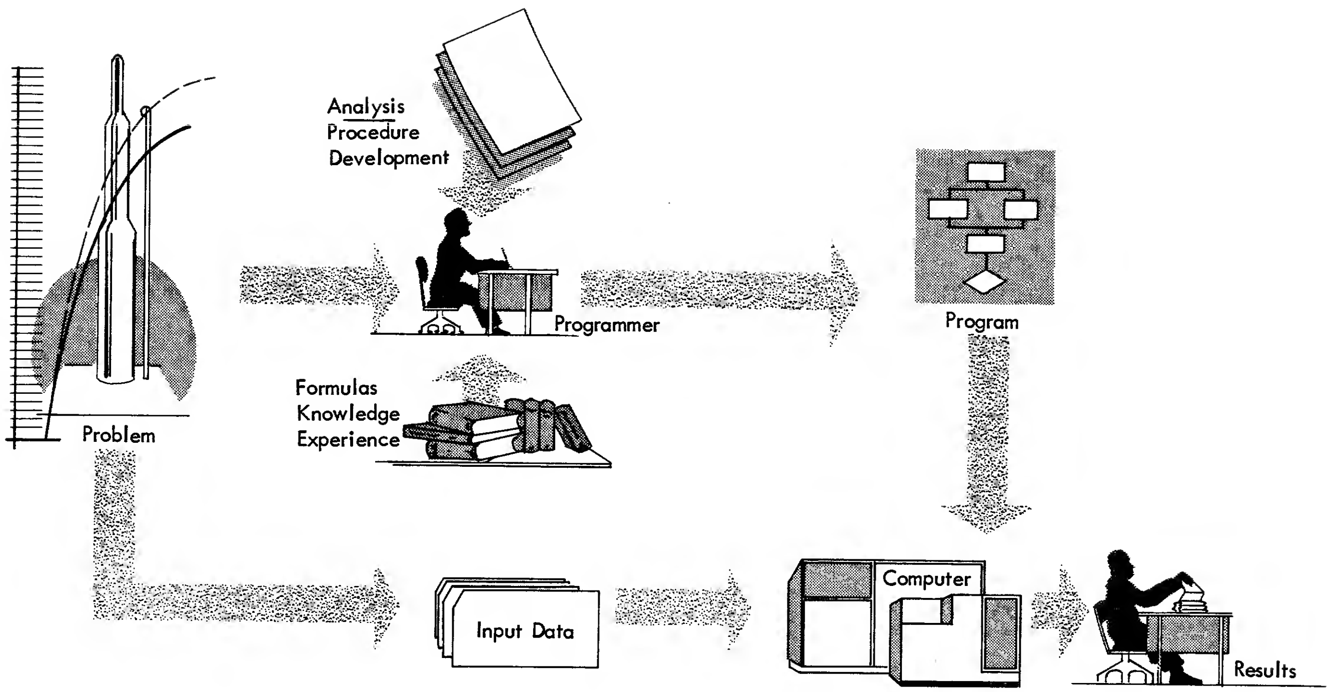


Figure 121. Direct conversion of problem to machine program

Types of Programming Languages

Many of the difficulties and inconveniences of writing programs directly in machine coding can be eliminated or simplified by the more advanced systems of program writing. The computer is used to prepare the programs and eliminate the need for direct machine coding.

A computer can be programmed to recognize instructions expressed or written in problem definition language and to translate those expressions into its internal machine language. This has led to the development of a number of programming languages that are easier to use and to understand than the language of the machine.

Symbolic Language

Symbolic languages permit the programmer to write convenient equivalents of machine instructions using symbols (called mnemonics) to represent them. Symbolic instruction representations include the following: A for add, S for subtract, D for divide, ST for store, B for branch, and so on. The computer, acting under control of previously written machine language programs, translates these symbolic instructions into equivalent machine instructions, which then can be used in solving the actual problem.

The first languages resulted in a one-for-one translation. That is, each instruction written in the programming language was translated into a single machine language instruction. For example:

A REG1,184

where REG1 is one of the 16 general purpose registers in S/360, would produce the IBM S/360 machine language instruction:

01011010000100000000000010111000

Later, "macro instructions" were developed. That is, single programmer language instructions could be used to produce a whole series of machine instructions. This development greatly increased the power of programming languages. The art of programming has progressed to a point where it is possible to give directions to a computer by writing statements and sentences in an English-like language that can be understood by both the computer and the programmer.

The translation feature of the machine language program is perhaps the most important feature, but not the only one. The computer instructions needed to produce a given result must be executed in a given sequence. If an addition is to be performed, one of the values involved must be in the accumulator before the add instruction itself is executed. This is normally accomplished with an operation called L, "Load". After this operation is executed, the add operation may be executed. The two-instruction sequence is shown in both a machine language and a symbolic language in Figure 122.

Each final machine language instruction must be assigned a particular location in core storage. For ex-

Machine	Symbolic
01011000000010000000000001100000000	L REG1, 384
01011010000010000000000001100001000	A REG1, 392

Figure 122. Machine and symbolic codes

ample, if the L instruction is to be assigned a location of 1000 (its precise location in core storage), and the add instruction is to immediately follow it, the location of the add instruction must be 1004 (since the load instruction is a four-byte instruction). Therefore, the location of each instruction must be known precisely. It is, in effect, the "name" of the instruction. If an additional instruction is to be inserted in a program of many instructions, every instruction from the point of insertion must have its previously assigned location changed. Since most programs undergo changing or updating, instruction location assignment becomes a tedious but necessary part of programming. The solution, of course, is to have the translating program do the actual assignment of instruction locations in addition to its translating function. The programmer need simply tell the translating program the desired location of the first instruction, and succeeding instructions are assigned sequentially ascending locations.

The advantage of expressing a problem in symbolic language over machine language should now be evident. This symbolism may be carried one step further by using symbolic data addresses as well as symbolic operation codes. The translating program can then be designed to translate and assign these symbols to actual core storage locations. Using the same instructions as before, assume that the two values to be added are expressed as values A and B. Of course, in both methods the values must have been previously placed in core storage, but the problem can now be stated as in Figure 123.

Instruction	
Operation Part	Address Part
L	REG1, A
A	REG1, B

Figure 123. Symbolic operations and addresses

If we now were to tell the translating/assigning program that we want the first instruction placed at core storage location 1000, the program shown in Figure 124 would result. (For better understanding, the program is expressed with symbolic operation codes and decimal addresses and locations, instead of machine language).

Instruction Location	Instruction	
	Operation Part	Address Part
1000	L	REG1, 4000
1004	A	REG1, 4004
.	.	.
.	.	.
4000	Value of A	
4004	Value of B	.
.	.	.
.	.	.

Figure 124. Assigned addresses and locations

The translating/assigning program is called a "translator or assembler". In normal operation, the translator is loaded into the computer system's core storage. Next, the instructions (prepared by the programmer to accomplish a particular job, as coded in his language) are entered into the computer. The computer then translates the programmer's instructions into machine language instructions, which use the order and logic set up by the programmer. The translated machine instructions are placed in core storage and form the actual program.

Language Translation

A programming language can be thought of in two parts: (1) the language itself, with associated rules of grammar, and (2) a machine language program (the translator), whose main function is to translate the language of the programmer into machine language.

The input to a translator is called the source program. This is written by the programmer in the language of the programming system (processor language), and states the requirements of the problem and the method of solution. Before the programmer writes his source program, he must have completely analyzed and defined the problem.

The output from the processor is the object program, the translation of the source program from the programmer's language to the language of the computer system on which the program will be used.

In some systems the object program may be executed. In others, the object program is not in executable format but must be processed by a program called the Linkage Editor which will produce an executable program. Subroutines (standard programs used with many problems), together with rate tables and other constant factors, may also be required within the computer to support the execution of the problem. The input or data source must be made available before execution can begin.

After the problem is executed (solved) by the computer, the result (output) may be recorded on magnetic tape or disk for later offline printing. Results also may be printed directly from the computer.

A proven object program may be used time after time, with varying problem data, to produce periodic results (such as production type programs of payroll or inventory) or to produce different results to assist the designer seeking an optimum design (such as the best wing airfoil or the most efficient placement of steam pipes within a boiler), considering all variables for each application.

Machine-Oriented Programming Languages

In a machine-oriented programming language, the programmer uses symbolic codes or names to designate operations that the computer is to perform. Symbols are also used to designate the location in main storage of data used with the operations. The translator then assembles the symbolic codes and translates them into machine language instructions with actual machine storage locations.

The System/360 Assembler Language is an example of a machine-oriented language. The programmer uses coding sheets (Figure 125) to write each instruction that the computer must perform. Each line of coding will be punched into one card. The vertical columns on the form correspond to card columns.

Space is provided on the form for program identification and instructions to keypunch operators. None of this information is punched into a card.

The body of the form is composed of two fields: the statement field (columns 1 through 71), and the identification-sequence field (columns 73 through 80).

Statements consist of from one to four entries in the statement field. From left to right, they are: name (eight characters), operation (five characters), operand and/or comments (56 characters).

The name entry is a symbol created by the programmer to identify a statement. A name entry is optional. It must consist of eight characters or fewer, and be entered with the first character appearing in column 1. No blanks may appear in the symbol.

The operation entry is the mnemonic operation code specifying the machine operation or assembler functions desired. An operation entry is mandatory. If there is no name entry, the operation entry may be placed anywhere to the right of column 1; if there is a name entry, at least one blank column must separate name from operation. (The same "free-form" rules apply to the other entries – that is, at least one blank must separate them, and they must be in the order described above, but otherwise they are not restricted in location.) To be able to see and comprehend easily what there is in the way of a program, it is best to follow the field column designations on the coding sheet and start the operation in column 10.

IBM — IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments
	TITLE	'ILLUSTRATIVE PROGRAM'	
PROGA	START	0	
BEGIN	BALR	11,0	
	USING	*,11	
	L	2,DATA	LOAD REGISTER 2
	A	2,CON	ADD 10
	SLA	2,1	THIS HAS EFFECT OF MULTIPLYING BY 2
	S	2,DATA+4	NOTE RELATIVE ADDRESSING
	ST	2,RESULT	
	L	6,BIN1	
	A	6,BIN2	
	CVD	6,DEC	CONVERT TO DECIMAL
	EOJ		END OF JOB MACRO
DATA	DC	F'25'	
	DC	F'15'	
CON	DC	F'10'	
RESULT	DS	F	
BIN1	DC	F'12'	
BIN2	DC	F'78'	
DEC	DS	D	
	END	BEGIN	

Figure 125. A program to illustrate assembler language concepts

Operand entries are the coding specifying and describing data to be acted upon by the instruction, by indicating such things as storage locations, storage area lengths, or types of data. Depending on the needs of the instruction, one or more operands may be written. The operands must be separated by commas, and no blanks may occur between the operands and the commas that separate them. Operands of machine instructions generally represent such things as storage locations, general registers, immediate data, or constant values. Operands of assembler instructions provide the information needed by the assembler program to perform the designated operation.

Comments entries are descriptive items of information about the program, usually something to remind the programmer of (or direct another programmer to) the purpose of the program step (or related sequence of steps). An entire line may be used for a comment by placing an asterisk in column 1. Extensive comments may be written by placing an asterisk at the beginning of each line.

There are many more conventions to the System/360 assembler language, but knowing this much about the makeup of the four types of entries should provide a basic understanding of the principles of coding.

Macro Instructions

The next step to increasing the effectiveness of a machine oriented language involves enlarging the functions of the translator.

The macro language provides the programmer with a convenient way to write a definition that can be used to generate a desired sequence of assembler language statements. Most macros are supplied by IBM. However, any user may develop his own to satisfy a special requirement.

The definition is written only once, and a single statement, the macro instruction statement, is written each time a programmer wants to generate the desired sequence of assembler language statements. This facility simplifies the coding of programs, reduces the chance of committing programming errors, and ensures that standard sequences of assembler language statements are used to accomplish desired functions.

The Macro Instruction Statement

A macro instruction statement (usually referred to simply as macro instruction) is a source program statement that can produce a variable number of machine instructions. Macros, just like assembler language statements, are source program statements that are processed by the assembler.

The assembler generates a sequence of assembler language statements for each occurrence of the same macro instruction. The generated statements are then processed like any other assembler language statement.

The Macro Definition

Before a macro instruction can be assembled, a macro definition must be available to the assembler. A macro definition is a set of statements that provide the assembler with (1) the mnemonic operation code and the format of the macro instruction, and (2) the sequence of statements that the assembler generates when the macro instruction appears in the source program.

The Macro Library

The same macro definition may be made available to more than one source program by placing the macro definition in the macro library. The macro library is a collection of macro definitions that can be used by all the assembler language programs in an installation.

Varying the Generated Statements

Each time a macro instruction appears in the source program, it is replaced by the same sequence of assembler language statements, unless one or more conditional assembly instructions appear in the macro definition. Conditional assembly instructions are used to vary the number and format of the generated statement.

Each problem-oriented language has its own method of writing macros. Some seem to be almost like writing plain English sentences. Problem-oriented languages include COBOL, FORTRAN, and PL/I, a new multipurpose programming language. They are called "problem-oriented" because they are tailored to the problem rather than to any particular machine.

COBOL System

With the COBOL (Common Business-Oriented Language) system the translator still must produce a machine language program before a problem can be solved. However, the language written by a COBOL programmer bears little resemblance to machine language, and the problem programmer has little direct concern with the method by which the COBOL language program is translated into machine language.

A simple example will best illustrate the basic principles of the problem-oriented type of programming system. Assume we wish to increase the value of an item called INCOME by the value of an item called DIVIDENDS. The COBOL language allows

us to specify the addition by writing the following sentence:

ADD DIVIDENDS TO INCOME.

Before the COBOL translator can interpret this sentence, however, it must be given certain information. For example, the programmer will have to write the names DIVIDENDS and INCOME in a special part of the program, called the "data division", where facts about the data represented by those names (such as maximum size, how the data is expressed, etc.) are stated.

When the translator encounters the sentence, it has access to certain information that will aid it in translating the sentence. In addition, it will be able to obtain certain information "built into" the translator itself. (Note, however, that the exact procedure will vary from machine to machine and that, in any case, the problem programmer is not directly concerned with the details.)

First, the translator examines the word ADD. It consults a special list of words that have clearly defined meanings in the COBOL language. This list is a part of the translator. If ADD is one of these words, the translator interprets it to mean that it must insert into the object program the machine instruction (or instructions) necessary to perform an addition.

The translator then examines the word DIVIDENDS. Since it can obtain certain information about DIVIDENDS, it will know where and how this information is to be stored in the computer, and it will insert into the object program the instructions needed to locate and obtain the data.

When the translator encounters the word TO, it again consults the special word list. In this case, it finds that TO directs it to the value of INCOME, which is to be increased as a result of the addition.

The translator must now examine the word INCOME. Again it has access to certain information about this word, and, as a result, it is able to place in the object program the instructions necessary in locating and using INCOME data.

We have indicated that the programmer placed a period (.) after the word INCOME, just as he would in terminating an English language sentence. The effect of the period on the COBOL translator is quite similar. It tells the translator that is has reached the last word to which the verb ADD applies.

The previously described steps are performed by the translator in creating the object program. They might not always be performed in exactly this way or in the same sequence, because machines vary and because each translator is adapted to a particular machine. However, regardless of the machine, the same COBOL language sentence produces machine instructions that cause the object program to add together the values DIVIDENDS and INCOME.

FORTRAN SYSTEM

The FORTRAN (Formula Translation) system is very similar in concept to the COBOL system. One of the main differences is in the language the programmer uses to express his source program. Where business English is used by COBOL, mathematical language is used with FORTRAN. The effect of the COBOL sentence

ADD DIVIDENDS TO INCOME.

could be achieved by the FORTRAN statement

INCOME = DIVIDENDS + INCOME

However, FORTRAN translators for some machines might insist that the words be abbreviated to something like:

INCO = DIV + INCO

This would depend on the individual machine FORTRAN translator. The statement, in effect, tells the translator to insert the necessary instructions into the object program to make the INCOME data location equal to the DIVIDEND data added to the present INCOME data. Note that the computer is not merely instructed to find the value of INCOME, but is also told where to put the result of the addition after it is performed. If the original INCOME field (in core storage) contained 10000, and the DIVIDEND field contained 15, the original INCOME field would be replaced by 10015 after the operation had been executed.

If this result is not desired, the programmer could change the statement to:

INCOME1 = DIVIDENDS + INCOME

With this change, a new INCOME1 data field would be generated in core storage, the result of the addition would be placed there, and the original INCOME field would remain unchanged.

PL/I System

The System/360 is able to handle problems from both the scientific and the business fields with equal facility. Neither FORTRAN (science-oriented) nor COBOL (business-oriented) allow access to the full capability of such a system as System/360.

Also, with the development of more and more decision-making, forecasting, and teleprocessing uses for computers, the business programmer requires frequent changes to his programs and a wider scope of computations. The scientific programmer, on the other hand, is faced with the task of handling problems with

masses of data, with a wider variety of input/output requirements. The differences between scientific and business programming are thus becoming less distinct. PL/I has been developed to meet the need for a broad-base language that may be used for both business and scientific applications. Some general characteristics and features of the language are summarized in the example in Figures 126 and 127.

In PL/I as in FORTRAN, addition (or other arithmetic operations) can be specified by the standard operators +, -, /, *, and **. For example, INCOME= DIVIDENDS + INCOME; PL/I differs from FORTRAN only in the presence of the semicolon (;) which ends a statement, and in that this statement can be written in free form, as found convenient by the programmer.

Addition can also be indicated by a statement which looks more like COBOL, such as:

SUM=ADD (DIVIDENDS, INCOME, 8, 2);.

This will put the result of adding DIVIDENDS to INCOME into an address named SUM which has an 8-position field, including two positions to the right of the decimal point.

ADD, DIVIDE, MUTIPLY, ATAN,COMPLEX, ERF, INDEX, MAX, MIN, ROUND, SUM, TAN, are

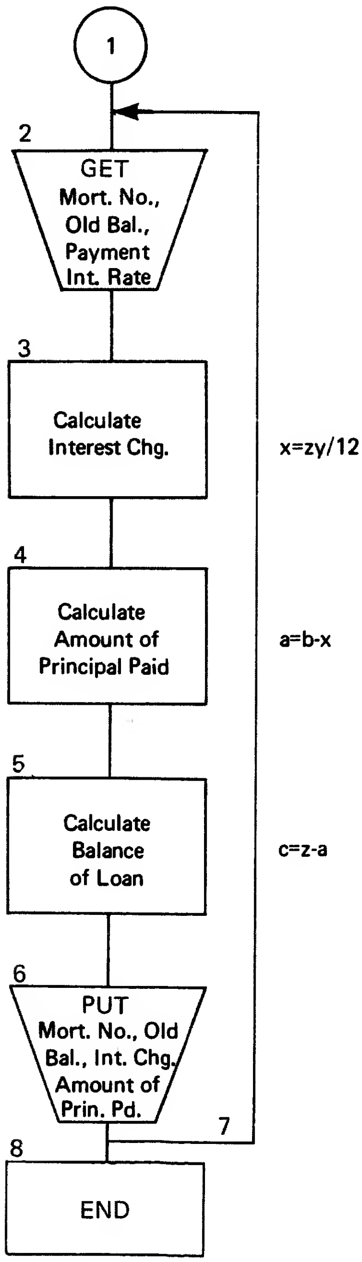


Figure 126. Flowchart of part of mortgage processing

```
BILLING:PROCEDURE OPTIONS (MAIN);
   NEXTCARD:GET DATA (MORNO,OBAL,PAYM,RATE);
                 CHARGE= OBAL*RATE/12;
                 PRINPAID= PAYM-CHARGE;
                 BALANCE = OBAL-PRINPAID;
              PUT DATA (MORNO,OBAL,CHARGE,PRINPAID,BALANCE;
   GO TO NEXTCARD;
END BILLING
```

```
BILLING:PROCEDURE OPTIONS (MAIN);NEXTCARD:GET DATA (MORNO, OBAL, PAYM,
RATE);CHARGE=OBAL*RATE/12;PRINPAID=PAYM-CHARGE;BALANCE=
OBAL-PRINPAID;PUT DATA (MORNO, OBAL, CHARGE, PRINPAID,
BALANCE) ;GO TO NEXTCARD;END BILLING;
```

```
BOB:PROCEDURE OPTIONS (MAIN);  /*THIS IS A COMMENT*/
    D:GET DATA (N,Z,B,Y);   /*COMMENTS MAY BE PUT*/
         X=Z*Y/12;    /*WHEREVER BLANKS ARE ALLOWED*/
         A=B-X;
         C=Z-A;
      PUT DATA (N,Z,X,A,C);
    GO TO D;
END BOB;


BOB:PROCEDURE;D:GET DATA (N,Z,B,Y);X=Z*Y/12;A=B-X;
C=Z-A;PUT DATA (N,Z,X,A,C);GO TO D;END BOB;
```

Figure 127. Coding of Figure 126 in PL/I

a few of the built-in functions available to the PL/I programmer to operate on individual data items, on arrays (vectors or matrices), or on structures (tables) of data items. The keywords of PL/I are not reserved words, and program text may be written in free form. For example, the program (called PROCEDURE in PL/I) on lines 1-8 in Figure 127 is the same as that on lines 9-12.

Keywords such as PAGE, SKIP, LINE, COLUMN, PAGE SIZE, LINE SIZE, make it possible to specify the format of printed output in as much detail as desired.

Report Program Generator

The Report Program Generator (RPG) language is a problem-oriented language. It is a very simple way of adapting an application to a computer. The file definition and input/output control considerations normally required of the programmer by other programming languages are reduced to the filling out of simplified control forms.

The input format specifications form is used to:

1. Specify the file or files to be read into the system.
2. Identify the different types of records contained in each file.
3. Describe the location of the data fields in each record.

The output format specifications form is used to:

1. Specify the kind of output files to be produced, printed reports, summary cards, etc.
2. Specify the location of the data fields of the reports or records.
3. Specify any headings or totals for printed reports.
4. Specify any editing or zero suppression needed for the printed reports or records.

The file description form provides additional information regarding the input/output files, such as specifying the input/output units used by the program and other features associated with input/output control.

The calculations form is the heart of the logic of the program. For calculation, RPG is a three-address language.

We can perform a mathematical operation (like add) on two addresses and store the result in a third address, all in the same statement.

The calculations form is divided into three categories, as follows:

1. Time to do the calculations.
2. Kind of calculations to be performed.
3. Tests to be made on the results.

Depending upon which System/360 programming system is used, there are 22-27 different operation codes. They include add, subtract, multiply, divide, table lookup, compare, move, branch, test zone, exit to a subroutine, etc.

Program Checkout

After successful translation of a source program, and then linkage editing in some systems, the next step in program preparation is to check the resultant machine language program by running it with test data. This is done to make sure that the program does not have logical errors and that it is capable of producing a right answer when using test data. Two results are possible. The first – and, hopefully, the only – result is that the problem (for which the program was written) can now be executed with real data. The second result – the test run does not function properly – may occur because of many things. The most frequent cause is that the source program has been improperly or incompletely stated.

Mistakes by the programmer are more difficult to avoid than might be expected. It is, in fact, a rare program that works correctly the first time it is tried with test data. In most cases, several test runs must be made before all mistakes are found and corrected. The translator itself finds most of the obvious mistakes during the translation run. Such things as calling for a storage location by a name when that name has not been defined, attempting to perform fixed-point arithmetic on floating-point data (or the reverse), lack of defined alternative paths on testing operations, and card-punching errors of all kinds are detected and noted during the translation run.

Computer mistakes are rare and usually obvious. Built-in detection circuits will normally reflect the kind of mistake the computer has made by turning on an indicator and stopping the computer. Detection and classification of the mistakes a programmer can make are, however, many times more complex.

Testing Techniques

As previously stated, a computer program may be expressed in machine, symbolic, or one of the problem-oriented languages, such as PL/I, FORTRAN, COBOL, or RPG.

Many techniques exist to assist the programmer during the checkout phase of his work. Each has its own advantages and disadvantages. The one to be used for a particular problem will depend upon the programmer's thoughts as to what area of his program is in error and how extensive the error is. Techniques that involve extensive use of switches on the operator's console are very wasteful of computer time and are not recommended. Two very useful techniques are storage printout and tracing.

Storage Printout

This type of utility program (routine) is helpful because practically the entire contents of storage, plus the contents of working computer registers and the condition of indicators and switches, may be presented in printed form. Normally, the register contents and the condition of indicators and switches are printed first. The contents of storage are then printed. Each line of printing representing storage begins with the starting location of that line expressed in octal or hexadecimal format. The print (dump) routine sometimes has provisions for dumping one or more selected blocks of storage instead of all of it.

Tracing

If visually checking a storage printout fails to reveal the program difficulty, a technique called tracing may be used. The trace technique usually involves an interpretive routine and, therefore, executes a number of instructions for each program instruction being traced. The printout received while tracing normally includes the location of the instruction being executed, the instruction being executed, and the contents of the working registers after the instruction has been executed. The printing of each instruction execution in a program would result in excessive machine time and should be used only when all other methods fail to reveal the program trouble.

The basic tracing technique may be revised so that only the contents of selected storage locations are printed when program execution reaches a specified point in the program. With this variation, a "snapshot" is obtained of a particular part of the program under particular conditions. For example, the trace and resultant printout may be specified to occur only when the program executes a transfer instruction. A whole series of snapshots then result showing the execution path through the program. Only the instructions that altered the normal execution path are recorded to show the exception paths the program has executed.

Summary

Successful program checkout depends on many things. The time consumed by this necessary but frustrating phase of programming may be lessened if certain basic rules are followed:

1. Document the program wherever possible, to enable anyone to know what a given program step is intended to accomplish.
2. Check the source program cards against the documentation before an assembly and test run is attempted. This point cannot be overemphasized.
3. Leave space in the program for insertion of testing or printing routines that may be used in the test run of the program. Program space is useful also if changes have to be made.
4. Be aware of the checkout techniques available, and know how best to use them; avoid becoming a slave to one technique, excluding all others.
5. Be absolutely sure that the program does what it is supposed to do and nothing more.
6. Be aware that a successful test run does not ensure that the program will run to completion with actual data. Actual data may be too large for the storage area assigned to it, too slow to be properly processed by the program, or not in the planned data format.

Input/Output Control Systems

While macro instructions save much labor, the problem of organizing input/output operations in a complex application could still involve considerable work on the part of even the most expert programmer. From a standpoint of simplicity, it is far easier to work with one record at a time—that is, read a record, process the record, and then write the resulting record—or to work with one scientific problem at a time. However, efficient use of tape or disk systems requires that records be grouped both on input and on output and that the processing of records be scheduled to best use the available computing time.

To solve this and other problems, the concept of input/output control systems (IOCS) was developed. Basically, adding IOCS to a programming language makes it possible for the programmer to think of his problem as a simple sequential operation. Given a description of how the input and output files are organized, the processor associated with the IOCS takes care of all the machine language coding necessary to read and write tape, card, or disk storage records.

The IOCS statements, which give the programmer the capability for input/output programming, are part of the total language for an individual system. An IOCS enables the programmer to divorce himself almost completely from the physical requirements of the data, the recording media on which the data is written, and the input/output devices on which the media are mounted; it permits him to concentrate most of his efforts upon the processing of the data.

With disk storage attached to data processing systems, additional complexities of input/output programming have been introduced. Because of the direct access nature of these devices, proper scheduling becomes even more important and more difficult.

Where a tape IOCS can use the serial nature of tape files to call in the next block from the tape before it is

requested by the user's program, this is not always possible when using a disk. Here, the "next" record of the file (data set) may be physically located anywhere in the disk storage, and several "logical" records may share the same physical disk record. Many techniques exist for solving this problem. Some are quite simple; others are very complicated. Several of the latter involve segmenting the user's program into two or more subprograms. Each subprogram can process one type of record or locate the new record of a given type. The IOCS can enter these subprograms in a more or less random sequence, depending on which of many records being sought on the file is found first. This is actually a simple form of multiprogramming, where several different logical programs perform their computation in a sequence partially dictated by a master scheduling routine.

Important features found in most of the input/output control systems in use are listed below. No one IOCS contains all features listed, but all of them use many of the procedures. Features not in universal use are included to show the great versatility of these systems. Input/output control systems have grown past the point of merely handling the normal input and output requests, and are becoming an integral part of the entire operating system for a data processing system, in some cases (as in Operating System/360), handling the manipulation of data internally as well as to or from the input/output devices.

Input/Output Scheduling

Some computers handle input/output in a serial, synchronous fashion. No computing can be done until an input/output operation is completed, and, conversely, no input/output can be done while the central processor is engaged in computation. Other computers, however, achieve simultaneous input/output and computing operation simply by allowing the central processor to continue with its operation while the input/output device locates data or reads data into or out of the main storage of the system. This simultaneous (asynchronous) input/output for all types of input/output equipment helps greatly to prevent unnecessary delay of the central processor while information is being read into or out of main storage.

An IOCS allows the programmer to easily make use of the complex asynchronous input/output devices that permit a modern data processing system to operate efficiently.

Blocking and Deblocking of Records

High-density tape or disk storage units become relatively inefficient when used to record short blocks of information. When recording 80 character blocks, for example, more than three-quarters of the file contains no useful information; instead, it is made up of end-of-block gaps. By grouping together or blocking a number of such short records, all but one of the interblock gaps can be eliminated. The result is that a given length of tape contains several times as much information as before. Since the tape passes through the tape unit at a fixed rate, the tape unit now spends more of its time reading useful information and less time spacing over gaps. The end result is a higher effective input/output data rate.

Note that the efficiency of the technique is reduced if the records are not requested often enough to keep the tape unit in continuous operation. In this case, the speed of the central processor becomes the limiting factor, and the program is said to be process-limited. If the reverse is true, the program is said to be input/output limited and blocking may be used to decrease the time required to read an average logical record.

Since input/output units usually require that the entire physical block be read or written once the function has started (there is no way to stop tape motion in the middle of a block), it is desirable to collect all records to be written as one block. Conversely, on input, it is desirable to unpack or deblock such a physical block into its many logical records and release them as requested by the processing program.

Standard Error Correction and Unusual Condition Routines

Many conditions met in performing input/output are exceptions to the normal case of simply reading or writing a record. The programmer does not wish to concern himself with all of these eventualities each time he makes an I/O request. For example, he should not have to perform a test each time a tape file (data set) is referenced to determine whether the end of the tape has been reached. Doing so makes the infrequent end-of-file condition require as much programming, perhaps, as the normal reading of the record. Many unusual or exceptional conditions are of a general nature and, as such, can be handled by common routines within an IOCS.

Listed below are a few of the exceptional conditions detected or handled by input/output control systems.

Error Correction Procedures

If transfer to or from an input/output device is not successful the first time it is attempted, certain techniques can be used to try to clear the failure and allow the program to continue without interruption. Such standard error correction routines might involve an attempt to erase a record recorded incorrectly on tape and to rewrite the record correctly. Obviously, if the

rewrite is successful, additional corrective instructions need not be used. Only if the repeated erase and rewrite are not successful in clearing the failure, does the machine operator or the program need to be informed of the uncorrectable error. Thus, most errors can be automatically corrected without additional programming being required.

End-of-Reel and End-of-File Procedures

When all data records on a single reel of tape are processed, the tape is said to be at end-of-reel. If, in addition, all records of the file, which can consist of more than one reel, are processed, the file is said to be at end-of-file. If an end-of-file condition is met, the processing of that file is complete, and the user's program must be informed of this fact.

Improper Length Record Procedures

If a record is read that, through malfunction or programming error, is not of correct length, this condition may be detected and corrective action taken.

If the error is such that the system cannot continue processing the current job, automatic transition to the next job can be initiated, or the system may be stopped after informing the operator of the nature of the error. In some cases, it is enough to inform the user's program of the condition and allow it to make the decision as to how the condition is to be handled.

Tape Labeling

The maintenance of a large library of tapes containing data costing thousands of dollars to generate imposes a large responsibility of preserving the integrity of the data. A careless operator, who inadvertently mounts a master tape containing valuable data (or removes file protection from a master reel by putting on a ring) and allows the tape to be written upon, can cause almost complete collapse of the application using this master tape (writing on tape automatically erases previously recorded information).

To ensure accurate library maintenance, a technique of tape labeling has been developed. This technique consists of recording, as the first information on each reel, one or more labels containing information that uniquely identifies the reel and also the information on the reel to the user's program or the IOCS label-checking routine. By comparing the desired reel and file identification against the information recorded on the reel, correct reel mounting can be ensured and file integrity preserved.

Disk Pack Labeling

Special utility programs or routines for disk labeling perform necessary labeling operations on the disk pack. The labels indentify the disk pack and the files on the pack. In some systems, another program sets up the track format.

Utility Programs

Utility programs involve certain functions common to all data processing installations. Two common types of utility programs are (1) those that tend to the details of converting a set of data from one recording medium to another (card to tape, tape to disk, etc.), and (2) record comparison programs. These and other utility programs developed by IBM and its customers are available as an aid to system operation. They serve a computer installation much as a book library serves an educational institution.

Data Libraries

The arrangement of, and access to, the data library is of paramount importance to any installation's smooth operation. Because mounting tapes and maintaining the library takes manual effort, time, care, and physical space, the trend, increasingly, is to store the library on disk packs, the larger disk storage devices, drums, and data cell drives.

In System/360, the input/output unit has a specific assigned address that cannot be changed. Within the input/output units, specific subunits – such as disk packs, data cells, tape reels, drums, and large disk access units – also have specific nonchanging addresses. Each of these subunits is called a volume.

The operating system itself can distribute incoming programs and other data into the volumes, keeping track of what is where and what space is available for new data.

Operating Systems

An operating system is an integrated set of processing programs and a control program designed to improve the total operating effectiveness of a computer. The operating system is directed by user-prepared control cards to pass automatically from job to job with a minimum of delay and operator intervention. Communication with the computer is via the operating system rather than directly, as was the case with most previous computers.

With today's more powerful machines, setup time is becoming proportionately larger for each job. As a result, large, expensive computers can sit idle at times while new work is being loaded. Even during the execution of a program, many components of the system may remain idle. Of all the resources available on a system, only parts of the total system may be required for a job. The operating system enables the user to stack jobs for continuous processing, thus reducing setup time between jobs. The operating system has the ability to call in any required programs, routines, or data. Some operating systems have the ability to schedule jobs and allocate resources more efficiently by concurrently processing two or more independent programs (multiprogramming). They can compile higher-level languages (COBOL, RPG, FORTRAN, PL/I), organize files, and act as a supervisor for the entire system.

In short, the operating system makes possible the maximum use of a computer.

In System/360, there are four different operating systems. Determining which is best for any particular S/360 depends upon the size of the system, the physical devices supported, and whether certain expanded functions are required, since features of one operating system may not necessarily be available in another. Briefly, the four operating systems are BOS/360 (Basic Operating System), DOS/360 (Disk Operating System), TOS/360 (Tape Operating System), and OS/360 (Operating System). Some of the similarities and differences in their makeup are discussed below.

What is intended is a brief survey of some of the major reasons for selecting one operating system over the others. A discussion of all supported devices and features is far beyond the scope of this manual. Omission of any particular feature is not meant to indicate lack of availability of that feature.

All four operating systems are composed of two major sets of programs, which are known as control programs and processing programs.

Control programs supervise the execution of the processing programs, control the location, storage, and retrieval of data, and schedule jobs for continuous processing.

Processing programs consist of language translators, service programs, and user-written problem programs, all of which the programmer uses to define the work that the computing system is to perform and to simplify program preparation.

System users may also include their own service programs or language translators, which the programmer can then use as he would IBM-written programs. Figure 128 illustrates the DOS/360, TOS/360, and OS/360 operating systems facilities.

In all four System/360 operating systems, the work to be done is regarded as a stack of jobs to be executed under the management of a control program. A job may contain one or more job steps, and may be thought of as a series of logically related programs that must be executed in a given sequence. Job steps within a job are always executed in sequence and never operate concurrently.

Necessary information about each job is punched into job control cards by the user. These cards are read by a portion of the control program to instruct it as to which programs are to be executed, and in what sequence; they also contain information about the I/O device requirements for each job step. The control cards provide the information necessary to operate in a stacked job environment.

BOS/360, TOS/360, and DOS/360 Control Programs

IPL Loader. This program loads the supervisor into main storage when system operation is initiated. The IPL loader is loaded from the system residence unit by dialing its address on the load-unit switches on the system console and pressing the load key.

Supervisor. This program handles all input/output operations, interruption conditions, and other control functions for all problem programs. Part of the supervisor resides in main storage at all times. Processing time is divided between the supervisor and the program being executed. This is true of user programs as well as the other IBM-supplied components of the system.

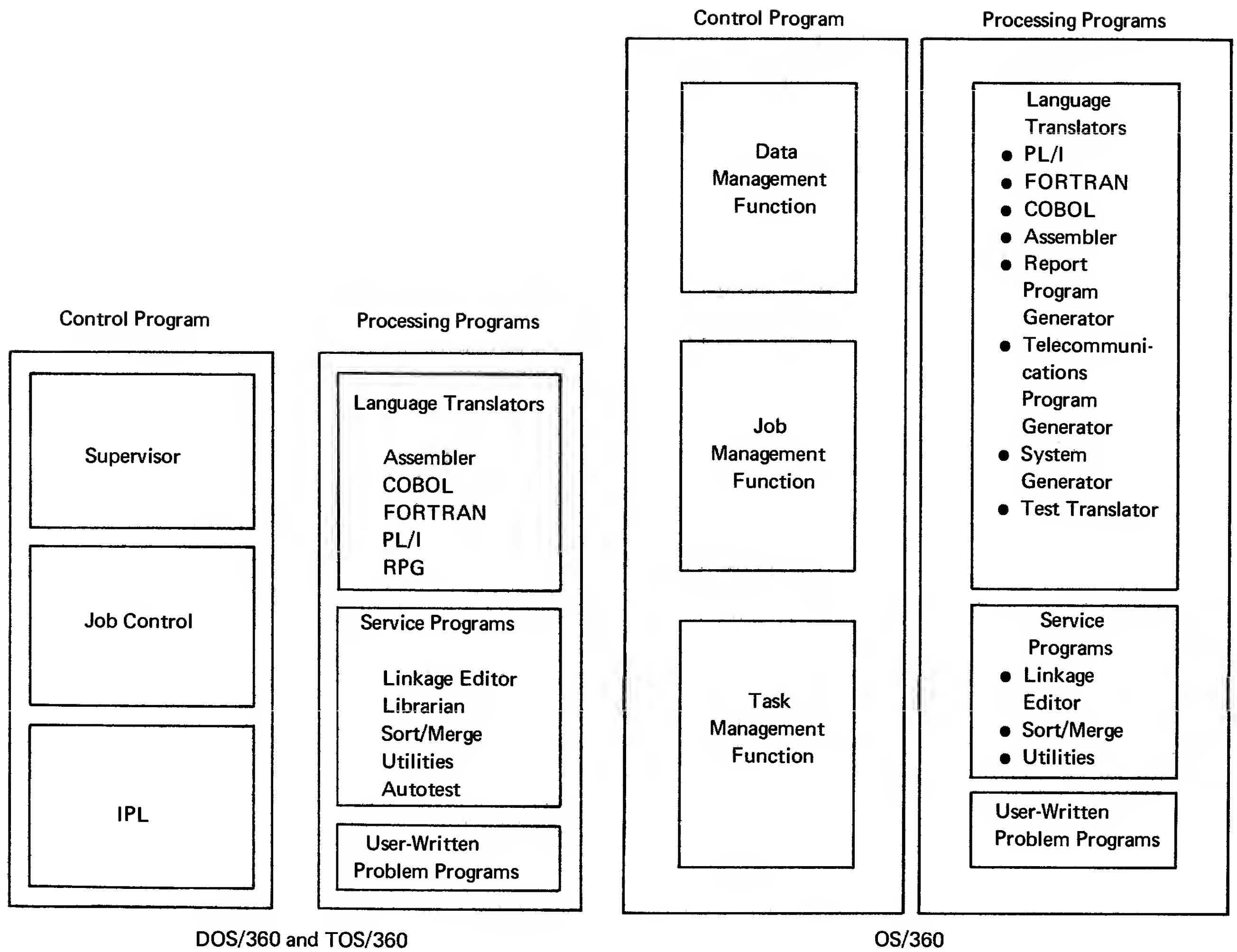


Figure 128. Facilities available in three of the System/360 operating systems

The physical IOCS routines of the supervisor handle the scheduling and supervise the execution of channel programs. The problem program (or logical IOCS within the problem program) supplies the channel programs and issues physical I/O macro instructions to request their execution.

The supervisor starts the I/O operation and returns control to the problem program. When the operation is completed, the supervisor checks for, and handles, any device error conditions. Thus, the user's program need not contain any I/O device error routines.

The checkpoint/restart routines of the supervisor provide a means of recording program status at desired points so that the program can be restarted there at a later time. The problem program resumes processing after each checkpoint. In response to a CHKPT (checkpoint) macro instruction, the checkpoint routine writes the problem program, along with other information needed to restart the program, onto magnetic tape. The restart routine can reload the program later and resume processing at the point of interruption. The restart program can reposition magnetic tape files before resuming program execution.

The supervisor transfers control of the system to job control at the end of each job step, providing transition between job steps and between jobs.

The storage print routine of the supervisor can provide a printout of core storage, and all registers, if an abnormal end-of-job condition occurs.

Some of the supervisor routines are loaded into main storage during system initialization. These routines are never overlaid and remain in main storage throughout execution of a stream of jobs. Other routines of the supervisor are called into main storage from external storage (tape or disk) only when their particular functions are needed. These are called transient routines. They are loaded into what is called a transient area, and they overlay any previous routines in the

area. This allows numerous supervisor functions to be provided while using a minimum amount of storage.

Job Control. This program runs between jobs and prepares the system for execution of all other programs; it is loaded by the supervisor whenever it is needed.

Logical IOCS

IBM furnishes a comprehensive set of macro definitions to create, access, and maintain data files. Descriptive macro instructions in the user's program generate the data and program logic for these files. (In BOS, the macro instructions must be assembled immediately preceding the rest of the problem program. Thus, they occupy an area of core storage between the user's program and the supervisor.)

Each imperative macro instruction issued by the programmer causes a branch to the proper instruction (in the generated IOCS logic) for the requested service.

Logical IOCS does the following:

- Requests physical I/O operations, as necessary, by issuing the required physical IOCS macro instructions. The necessary channel programs are generated from the descriptive IOCS macro instructions.
- Supplies logical input records to, or accepts logical output records from, the problem program. This includes blocking and deblocking logical data records (fixed-length or variable-length) from larger physical blocks. (Logical record refers to the individual unit of a data file; physical record refers to the block of logical records read or written as a single string of information.)
- Switches between two I/O areas to provide time for processing while records are being read or written.
- Handles end-of-file and end-of-volume conditions.
- Constructs and maintains file organization structures. This includes additions and deletions to files, and the construction and use of index tables for processing files.

Label Processing

Disk and tape label-processing capabilities are included to provide:

1. Assurance that the correct editions of disk and tape data files are provided for input and (in the case of multipack or multireel files) that this input is provided in the correct sequence.
2. Assurance that areas of disk storage or tape reels designated for output contain no current information. If usable, new labels are written for the output areas or reels.

The actual label processing is performed by transient routines assembled as part of the supervisor during initial system generation. These routines are loaded into the transient area of the supervisor and executed in response to macro instructions in the problem program. TOS/360 has only tape label-processing routines, whereas BOS/360 and DOS/360 have both tape and disk routines.

OS/360 Control Programs

Like BOS/360, TOS/360, and DOS/360, OS/360 has an IPL loader to initialize the system.

Data Management

This facility handles all IOCS functions relating to tape, disk, drum, and telecommunication activities. It can build program libraries and extract programs as needed. The programmer calls for all data in essentially the same way.

In addition, data management provides:

- Allocation of space on DASD.
- Automatic location of data sets.
- Protection of data sets against such occurrences as unauthorized access to security files (for example, payroll information may require "passwords"), an accidental attempt to write over a file that is to be saved, concurrent updates of the same record in a multiprogramming environment, etc.

Job Management

This facility handles the scheduling of jobs to be done in a stacked job environment. Two types of scheduling are possible: the sequential scheduler initiates jobs on a first-in, first-out basis; the priority scheduler initiates jobs on the basis of user-assigned priorities.

Task Management

This facility controls the operation of the system as it executes tasks. A job step (program) becomes a task when its I/O devices have been allocated, and it is known to the control program as a unit of work to be done. The task supervisor, responsible for interrupt handling, program fetching, storage contents management, etc., may handle a single task, a fixed number of tasks, or a variable number of tasks, depending upon the version of OS/360 being used. If many tasks are contending for use of the CPU resource, the task supervisor determines which task is to gain control.

Processing Programs

Each operating system discussed has, in addition to the control program, many other programs that are, for the sake of simplicity, classified as processing programs. To further break down this classification, there are: language translators, service programs, and user-written programs.

Language Translators

A language translator is defined as a routine that accepts statements in a programming language and produces equivalent statements in machine language.

BOS/360 supports the S/360 Assembler Language, which includes a complete set of macro instructions. The S/360 Assembler Language is a machine-oriented symbolic language used in all models of System/360.

In addition, BOS/360 also includes a Report Program Generator (RPG). RPG is a problem-oriented language designed specifically for report-writing and file-maintenance applications.

DOS/360 and TOS/360 have, in addition to the assembler and RPG, COBOL, FORTRAN, and PL/I language translators.

OS/360 has, in addition to all of the above-mentioned language translators, an ALGOL translator.

The source languages that are provided for System/360 are upward-compatible. For example, if a user has a source FORTRAN program written according to DOS/360 or TOS/360 specifications, he can use the same source program with OS/360.

Service Programs

The system service programs for BOS/360, TOS/360, and DOS/360 include linkage editor, librarian, sort/merge, utilities and load-system (BOS/360 only).

Linkage Editor

The linkage editor links separately compiled decks, relocates these decks as required, resolves external references, and includes modules from the relocatable library as necessary.

All programs are edited onto the resident disk (DOS/360) or a utility tape (TOS/360) by the linkage editor. These programs can then be permanently placed in the core image library of the system, requiring only control statements to call them for execution. Alternately, they can be executed at once and then overlaid by new programs.

Librarian

This group of programs maintains the libraries and provides printed and punched output from them. Three libraries, all residing on tape or disk, are available:

1. Core Image Library. All programs cataloged in the system (IBM-supplied and user programs) are loaded from this library by the system-loader routine of the supervisor.
2. Source Statement Library (DOS/360, TOS/360). This library contains IBM-supplied and user-defined source statement books, such as macro definitions. A book is an arbitrary collection of 80-byte records that is cataloged under a single name in the source statement library.
   Books are maintained in compressed format on the resident volume to conserve space and improve their speed of retrieval. Complete books may be added or deleted from the library (but not individual records). These books can be copied, for example, into assembly source programs or COBOL source programs.
   BOS/360 does not have the full source statement library facilities. Rather, its macro library stores IBM-supplied and user-defined macro definitions in resident packs built to provide program assembly capability.
3. Relocatable Library. This library stores object modules for later linkage into complete programs.

Load-System (BOS/360 only)

This is an independent program that is loaded from cards. It has its own initial program loader (IPL), supervisor, and job control programs. The load-program builds a resident system from cards. This program can be used to build minimum systems for specialized applications. If two disk drives are available, the librarian can be used instead of the load-system program to build specialized systems.

Sort/Merge Programs BOS, TOS, DOS

The IBM BOS/360, TOS/360, and DOS/360 sort/merge programs provide the user with the ability to sort files of random records, or merge multiple files of sequenced records, into one sequential file. The control data information can be contained in as many as twelve fields in each record. The records can be sorted or merged into ascending or descending sequence. An individual sequence can be specified for each control data field. The output sequence for a merge-only operation must be the same as the input sequence.

The sort/merge program is a set of generalized modules (in the relocatable library) that must be tailored at execution time to each application. The user furnishes appropriate parameters on control cards; the tailored sort/merge program is built in the core image library; and it is then automatically executed as a sequence of overlays from this library.

BOS/360 and DOS/360 furnish disk sort/merge programs; both DOS/360 and TOS/360 furnish tape sort/merge programs.

OS/360 Sort/Merge and other Service Programs

The facilities available in OS/360 with respect to service and other sort/merge programs are basically the same as in DOS/360. However, they are an expanded set of service programs with far fewer restrictions than in DOS/360. For example, the DOS/360 sort/merge programs have extra restrictions as to size and number of files, whereas OS/360 sort/merge programs are more liberal in this respect. In addition, OS/360 sort/merge is faster and more versatile.

OS/360 does not have a librarian *per se*, as does DOS/360. However, a set of utilities is available for maintaining all data sets. In addition, OS/360 allows the user to supply his own programs or macro instructions for maintaining data sets. Figure 129 shows program stages in DOS/360 and OS/360.

Utility Programs

BOS/360 contains eleven file-to-file utility programs:

- Tape to tape.
- Tape to disk.
- Tape to card.
- Tape to printer.
- Disk to tape.
- Disk to disk.
- Disk to card.
- Disk to printer.
- Card to tape.
- Card to disk.
- Card to printer and/or punch.

The initializing utilities in BOS/360 are:

- Clear disk, which clears as little as one track up to one disk pack.
- Tape compare, which compares two files from two or more tapes to ensure that the files are identical.

TOS/360 has the same utilities as BOS/360, with the exception of those utilities involving disk. DOS/360 has all of the BOS/360 utilities, plus six addi-

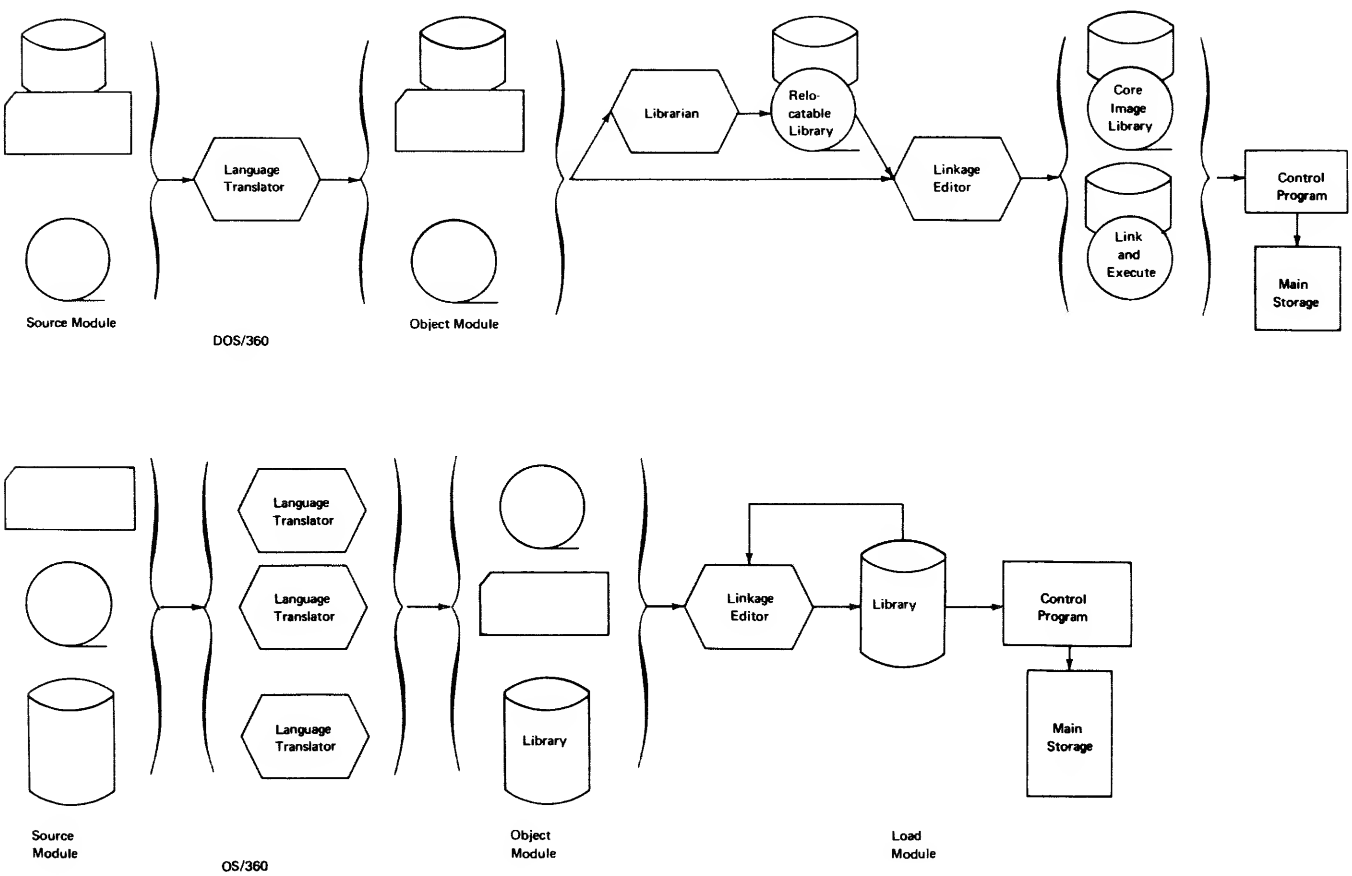


Figure 129. Program stages in DOS/360 and OS/360

tional utility programs for the 2321 data cell. The additional utility programs are:

- Tape to data cell.
- Disk to data cell.
- Data cell to tape.
- Data cell to disk.
- Data cell to data cell.
- Data cell to printer.

In BOS/360, TOS/360, and DOS/360, utility programs are device-dependent (such as tape to tape, etc.). In OS/360, utilities are device-independent because the utilities refer only to data sets rather than to devices. Otherwise, they operate in the same fashion as the utilities in DOS/360. OS/360 has the ability to use the following direct access storage devices:

- 2301 drum.
- 2302 disk storage.
- 2303 drum.
- 2311 disk drive.
- 2314 direct access facility.
- 2321 data cell.

Testing and Debugging Programs

A programmer who has for his use a set of test service routines can be relieved of much time and frustration involved in debugging. Such a set of routines is available for System/360 operating systems. In BOS/360, TOS/360, and DOS/360, it is known as Autotest. In OS/360, it is known as Testran. Both perform services at certain points in the program (specified by the programmer). These test actions include dumping (recording for display) systems tables, registers, and main storage; also, tracing of transfers, calls for subroutines, and references to data. Consecutive tests can be run with different sets of data. They can also test conditions stemming from program execution and, according to the results, either carry out or not carry out the dumps, traces, etc.

User-Written Programs

Computer users generate many programs to solve a specific set of problems. Some of the routines may be useful to other users in solving similar problems. The widespread incorporation of routines that have been written elsewhere makes well-defined programming conventions especially important. Such standards are necessary if programs are to be interchanged among computer users.

Agreement on standards has been facilitated by the development of organizations such as SHARE, GUIDE, and COMMON. Group members exchange programs through program libraries established by these groups.

Teleprocessing

Teleprocessing capabilities are available in DOS/360 and OS/360. Both DOS/360 and OS/360 are basically the same. They may be used for a large variety of teleprocessing applications, including the following:

- Message switching. Messages received from one remote terminal are sent to one or more terminals.
- Remote job processing. Jobs received from remote locations are routed and assembled for processing by the job scheduler.
- Inquiry or transaction processing. Inquiries or transactions received from a large number of widely separated locations are routed for processing by user-supplied message-processing programs.
- Data collection. Data received from a number of terminals is collected and stored for later processing.

The three major levels of access methods supported in the telecommunications area are BTAM (Basic Telecommunication Access Method), QTAM (Queued Telecommunication Access Method), and STRAM (Synchronous Transmit Receive Access Method).

BTAM provides for polling and addressing of terminals, and for sending and receiving of messages. The user-provided programs must include the following:

1. Initiation of polling and addressing (macro instructions must be given to start the process).
2. Routing of messages to the proper destination.
3. Error checking of messages.
4. Code translation, label checking, and queuing and logging of messages, if required.
5. Processing of the data content of the message.

QTAM provides for all of the above services except processing of the data content of the message.

Both BTAM and QTAM are for use with start/stop type terminals, such as the IBM 1050, and can handle transmission speeds of about 15 characters per second.

STRAM provides macro instructions and routines to allow transmission and reception of data through synchronous transmit/receive terminals. STRAM has the following features:

1. Line control.
2. Environment definition.
3. Code conversion.
4. Data transmission.
5. Error recovery.

STRAM is used with the synchronous transmit/receive (STR) terminals, such as the IBM 1009 and 1013, which have faster transmission rates. For example, the 1009 Data Transmission Unit can transmit up to 600 characters per second.

Compatibility and Emulation

In the past, a change in computers usually entailed a costly rewriting of programs originally developed for the system being replaced. To protect programming investments of users selecting S/360 to replace previously installed IBM systems, IBM has developed equipment features and programs designed to ease transition.

Compatibility features on S/360 enable a user to take non-S/360 programs, which adhere to acceptable coding standards for their original machine, and, without change, execute them on certain specified models of S/360. For this purpose, a device called Read-Only Storage (ROS) is used.

ROS is a device built into S/360 that is a step below machine instructions (just as machine instructions are a step below FORTRAN instructions). This concept is called microprogramming. Instead of executing an instruction, ROS executes a series of openings and closings of electronic "gates" in the system data flow. Figure 130 shows an ROS card. Each line of perforations controls the data flow for one instruction of machine language. Most models of S/360 have ROS for S/360 instructions, but special ROS cards must be used for compatibility. For programs written in 1401/1440/1460 or 1620 machine language, 1401 or 1620 ROS converts them into a series of microinstructions executable on S/360.

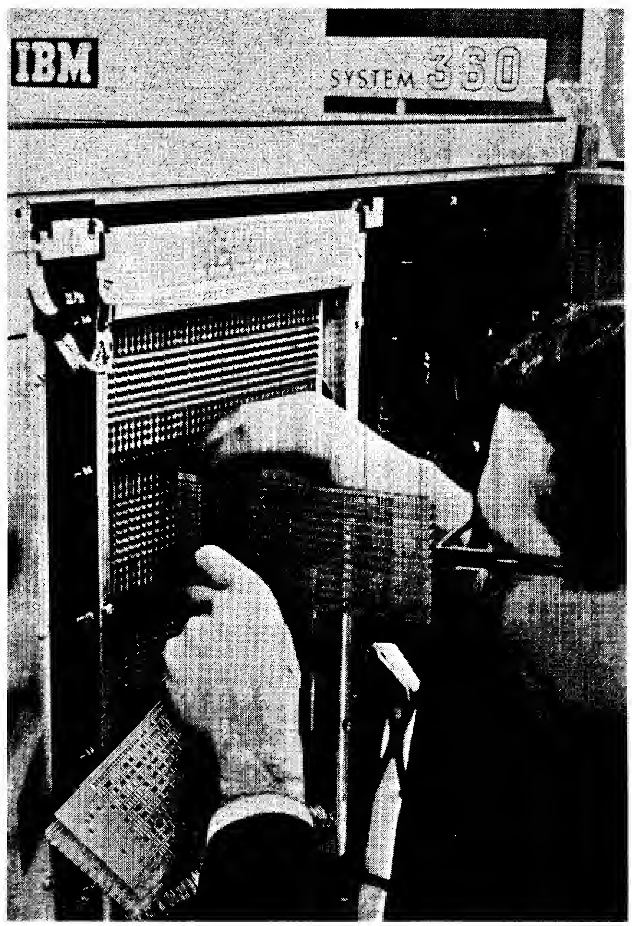
Emulation involves the linking of a compatibility feature (ROS) and a core-resident S/360 program (called an emulator program) to execute a non-S/360 program that is also in main storage. The program is under control of ROS until the compatibility feature determines it cannot handle a certain operation, at which time control is passed to the emulator program, where the particular operation is executed in S/360 mode. At the completion of the operation, control is passed back to the compatibility feature for the next instruction. At present, there are emulators for several IBM computers that preceded System/360, including:

Native Machine	S/360 Model Required
1620	Model 30
1401/1440/1460 1410/7010	Model 40
7070/7074 1410/7010	Model 50
705/7080 709/7090/7094 7040/7044	Model 65

Some considerations for emulation are:

- The amount of core required by the program. (The amount required on an emulating machine is always greater than the storage capacity of the native machine.)
- Device and operation code restrictions, if any.

Although emulation and compatibility enable users to run their programs from a previous machine on S/360 with at least equal effectiveness as on the previous system, the performance will not take advantage of the full capabilities of S/360.



A major element in converting programs for use in IBM System/360 is this special perforated card being inserted into the read-only storage unit of the Model 30. The read-only storage unit contains many of these instruction-bearing cards that enable programs written for an IBM 1401 to be run on the Model 30 without change.

Figure 130. Read-only storage unit and card

A data processing procedure must include two main areas of activity: accomplishment of the desired result and control of the procedure itself. Complete controls are far broader than the checks designed to supervise the quality of work produced by the computer system. These controls must also consider the entire application and its importance in a business or scientific endeavor.

Methods must be devised to control the flow of information into and out of the data processing system and to assure that all information received is correctly included in the required results. In addition, should omission or duplication occur, methods must be devised to establish an audit trail without completely retracing an entire procedure.

Control procedures differ somewhat between business applications and engineering, statistical, scientific, or mathematical applications, though both types may exist in one installation. In scientific applications, principal control is exercised on the accuracy and range of calculation only, and, while control of data must also be strict, the requirements of auditing are usually simple or nonexistent.

On the other hand, business records are the property of stockholders, and, as such, they must be available for both external and internal auditing. Records must also be protected from the possibility of fraudulent practices and must legally conform to tax structures, public service codes, and other local and regional restrictions. In many businesses, the method of record-keeping directly affects relations with customers, and files must be accessible for inquiries and account status, position of inventory, availability of services, and so on.

The following discussion is concerned primarily with the area of business practices. In any application, however, the purposes of overall procedure control are to:

1. Assure that data entering the computer is accurate.
2. Check clerical handling of data before it reaches the computer to assure that it is complete and not duplicated.
3. Arrange data in the form best suited for economical use by the computer.
4. Provide a means of auditing the steps of the complete procedure so that, in the event of error or inconsistency, the trouble may be located with minimum loss of time.
5. Assure that accounting for tax purposes or other legal requirements is carried out according to law.
6. Guard against fraudulent practices that may affect the financial standing or reputation of the business.

Control Groups

The data processing center is usually a service unit. It receives information to be processed (perhaps in the form of punched cards) from outside sources, makes the necessary calculations, and produces the necessary reports. The center, as a rule, originates no information; it is concerned only with data sent to it. Under these circumstances, it is possible to establish controls over the employees of the center and to prevent fraudulent or inaccurate handling. These controls are usually delegated to some group, either organized within the center or closely associated with it for this purpose.

Control of Payroll Data

One large company computing its payroll with an electronic installation established an office known as the payroll bureau. All changes in payroll data—such as rate changes, new employees hired, terminations, and changes in 25 types of pay deductions—are routed through this bureau. The changes are indicated on an authorization form, prepared in duplicate in the originating department.

The original approved copy of this document is forwarded to the payroll bureau, where the change authorization forms are grouped by type of pay data affected. Adding machine totals of numeric fields are accumulated, regardless of whether these fields represent dollar information or identification; in the case of new employees or terminations, the number of employees affected is also included in these control totals. Cards are then punched and key-verified for input to the computer.

The totals of the changes are accumulated weekly from the cards on a conventional punched card accounting machine. These totals are checked with the adding machine totals previously prepared. During one of the computer runs, normal pay and pay deductions are calculated. This sum is sent to the payroll bureau, where the totals are compared with those previously recorded. This control over payroll changes

not only incidentally checks the operation of the data processing system but also checks the clerical handling and accumulation of information as it enters the system.

Control of Sales Data

Procedures can be controlled to assure the best relations with customers. One example of this control is supervision of the company's price structure for the commodities or services it sells. Here, a control group might be established to compute amounts to be charged to a customer.

In many organizations, for example, the sales department has discretion over the prices that customers are charged. However, deviations from established prices may occur because of allowances for defective merchandise or because of special situations. The function of the control group is to see that all exceptions from the official price list are investigated and can be explained. The computer is used to report these exceptions.

The identification code of merchandise shipped and the code number of the customers are entered into the computer, along with special notification if the sales price differs from that shown in the master price file. The computer prices all shipments, except those for which it has special notification, at the master price rate and follows special instructions for the exceptions. When the output data set (or file) that will eventually print the invoices is produced, a special listing is made of all shipments that have been calculated at other than master file prices. The listing is forwarded to the control group for investigation.

The control group may have other control functions that relate to the contents of the master files. An electronic system differs significantly from a manual system in the method of referring to the authorized selling price information. Where invoices are computed by billing clerks, it is reasonable to assume that, while individual clerical errors may occur, original price information is accurate. This is so because, as a rule, the clerks use an identical, up-to-date price list.

In the computer system, on the other hand, the prices are inserted from a master price data set (or file), and reference is made to this data set by the machine in determining the billing price. If an error is made in the price of a particular product as recorded in the master data set, this error is reflected in the billings for all customers purchasing this item. Therefore, the contents of the master data set must be strictly controlled.

Changes to the data set may be made during a special run by the computer. Price changes are inserted, and a change register is produced. A copy of the change register is forwarded to the control group, where it can be examined in detail. From time to time, the master data set can be used to prepare a complete printout of portions or all of the product master data sets. It may also be used, if descriptive information is present, to prepare a price catalog.

Similar controls can be established to make certain that shipment is made only to customers of acceptable credit standing. This control can be established by a customer master data set that is used to extract the proper shipping and billing addresses and to carry the approved credit limits. By proper control, it can be determined that shipments are made only to customers whose credit standing is approved by the credit department. Companies doing business with franchised dealers can use this procedure to detect shipments to unauthorized dealers, because the absence of a name in the customer master data set immediately prevents invoice preparation.

Systems Checks

Systems checks are designed to control the overall operation of a procedure within the computer system. They ensure that all required data is received for processing and that all data leaving the system is complete and accurate.

Systems checks may include controls to ensure that all input records are included in a data set for a current processing period and that incorrect or unrelated records are excluded. These checks may also verify the distribution of detail transactions to update records when such distribution is made by coding. Systems checks may be devised for factors developed during calculation to compare this logical or reasonable relationship with other known factors. Crossfooting totals is a commonly used systems check to prove calculation or accumulation of quantities and values.

The types of systems checks vary with each application of the computer and with the kind of equipment used. Particular attention should be paid to including systems checks during the early stages of application planning, because such controls can be most effectively fitted into the program at that time. It is sometimes advisable to modify the procedure to include the most efficient controls; this is usually far less costly than designing a procedure without required controls.

Many commercial procedures require strict accounting control with provision for audit trails. This required control means that the program must be designed to take full advantage of the high reliability built into the data processing system. To meet the

requirements of efficient operation of the entire system, this control also means that the trouble can be localized quickly in case of error, without retracing the entire procedure or reprocessing all records.

In some machines, built-in checking features make detailed systems checks unnecessary. In others, data manipulation in the central processing unit is less stringently checked, particularly where elaborate checking circuitry would materially increase the cost of the system without a proportionate increase in accuracy. Some systems checks are supplied as part of programming systems; for example, in IOCS.

Checking is a form of quality control. It follows that, when some percentage of error can be tolerated, checks may be used more sparingly. Some typical systems checks are discussed below.

Record Count

A record count is a tally of the number of records in a data set. The count is normally established when the data set is assembled.

The total number of records is carried as a control total at the end (or the beginning) of the file (data set) and is adjusted whenever records are added or deleted. Each time the data set is processed, the records are recounted, and the quantity is balanced against the original or adjusted total. If the recount agrees with the control total, it is accepted as proof that all records have been run.

Record counts may also be established by batches. This is desirable when source data is to be put into the procedure for the first time.

Although the record count is useful as a proof of processing, it is difficult to determine the cause of error if the controls are out of balance. A failure to balance does not help to locate a missing record, nor does it indicate which record has been processed more than once. Therefore, some provision must be made to check the data set against the source records, a duplicate data set, or a listing known to contain the proper number of records.

An incorrect record count usually indicates a machine failure when records are being processed because, once written on the tape (or drum, disk, etc.) correctly, records cannot be misplaced or lost. In this case, the doubtful portion of the data set should be rerun for correction. An IOCS supplies information as to the bad record.

Control Total

The control total may be made up from amount or quantity fields in a group of records. It is accumulated manually or by machine when the data set is originated or when a quantity is first calculated. The control total can be either a grand total or more convenient intermediate or minor totals.

When the data set or group of records is processed, the fields are again accumulated and balanced against the control total. If the total is in balance, it serves as proof that all records have been processed correctly.

The control total is an efficient systems check when it can be used to predetermine the results of calculation or the updating of some record. For example, when preparing to process a payroll, the total number of hours worked by all employees is prestablished from clock or job-card records. This figure then becomes the control total for payroll hours for all subsequent reports. Totals may be broken down by group or department. The sum of all totals must balance back to the complete original total.

Control totals are normally established for batches of convenient size, such as department, location, account, or division. By this method, each group of records may be balanced as it is processed. Corrective action, if needed, is limited to small, easily checked groups rather than to one grand total.

Proof Figures

Proof figures may be used to check an important multiplication in a procedure. As such a check, the proof figure becomes both a systems check and a check on the operation of the computer. The proof figure is usually additional information carried in the record.

An example of the proof figure is the multiplication of quantity by unit cost. The check is based on the relationship between actual unit cost and a so-called proof cost. An arbitrary fixed figure (Z) larger than any normal cost is set up. (If a cost range for all products in a given data set is from $.50 to $10.95, Z might equal $11.00). Proof cost is the remainder when cost is subtracted from Z, or proof cost may be expressed by the formula:

$$\text{Cost} + \text{Proof Cost} = Z$$

Proof cost is carried as an extra factor in each record. Z is a constant that can be placed in storage for use when the proof figure is calculated.

Whenever quantity is multiplied by cost, it is also multiplied by proof cost. Normally, the factors accumulated during processing are quantity, quantity x cost, and quantity x proof cost. At any point, it is possible to check the sums of all factors accumulated up to this point as follows:

$$\Sigma(\text{Qty} \times \text{cost}) + \Sigma(\text{Qty} \times \text{Proof Cost}) = \Sigma(\text{Qty} \times Z)$$

The left side of the equation can be calculated by a single addition of the two progressive totals that have been accumulated during the procedure. The right side of the equation can be calculated by a multipli-

cation of the accumulated quantity and the constant factor Z. This check ensures that each particular multiplication was performed correctly.

This type of proof figure can be applied to other applications by using the same general approach.

Limit Check

A limit check is a test of record fields or programmed totals to establish whether certain predetermined limits have been exceeded. For example, if transaction codes for certain records are known to cover only the digits 0 through 5, a check can be programmed to see that no code exceeds the limit 5.

A second type of limit check assures that calculated totals are reasonable. Some quantities or values in a procedure never vary more than a given percentage between processing periods.

Payroll procedures often contain many limiting factors that can be checked by the program. The upper limit of gross pay is usually determined by the type of payroll: hourly, salary, piece rate, incentive, and so on. Hourly rates must fall within established wage scales. The total number of hours worked per employee is also subject to certain limits.

Limit checks may also be used in table lookup procedures. If an item is known to be in a given table in storage, the modified table address may be checked against the address of the upper table limit to verify correctness of the search. If the search begins to exceed the limits of the table, an error has occurred, and corrective action is required.

In many mathematical problems, the range of the final calculation can generally be estimated. If a result falls outside this reasonable range, it may be assumed that some error condition is present, either in the data, in the program, or in the calculation. Departures from normal trends may also indicate faulty procedures. The simple application of a limit check in such problems may save much detailed checking, with consequent simplification of the program.

Crossfooting Checks

Crossfooting checks may be used to check known control totals, or they may serve as proof totals originated during a procedure. For example, during the processing of employee records in a payroll, calculations develop amounts of gross pay, taxes, deductions, and net pay. Normally, these amounts are accumulated by department or other convenient batch controls. The totals of gross pay at any point should be equal to the totals of net pay, deductions, and taxes.

Tape and Disk Labels

Identification information recorded at the beginning of a reel of magnetic tape is called the header label; identification information recorded at the end of a reel is called the trailer label. The label may specify file identification, date of last processing, number of reel, and so on. A label may also be placed at the end of the file. Standard label checking is automatic in programming systems.

While tapes may have both a header and a trailer label located physically before and after the data, respectively, disk labels can appear physically anywhere on the volume, as long as the user specifies where it is located.

The labels are read into storage at the beginning and at the end of the program as an added control to ensure that the proper records have been processed. The label may also ensure a true end-of-file or end-of-job condition and, in addition, include a record count.

The various programming systems supporting System/360 have a definite format for "volume" labels and checking.

Housekeeping Checks

The first instructions of nearly every program are intended to perform functions of housekeeping in preparation for processing. These instructions may set program switches, clear registers, set up print areas, move constants, and so on. In addition, housekeeping instructions may perform systems checks by testing to determine whether all input/output units required by the main program are attached to the system and ready for operation. File labels may be checked and updated, constant factors may be calculated, and other information pertinent to the proper operation of the system may be called to the operator's attention by programmed instructions. Programming systems provide many of these checking procedures.

Checkpoint and Restart

A checkpoint procedure is a programmed checking routine performed at specific processing intervals or checkpoints (see "Program Checkout"). Its purpose is to determine that processing has been performed correctly up to some designated point. If processing is correct, the status of the machine is recorded, usually by writing this information on a tape. The normal procedure is then continued until the next checkpoint is reached.

Checkpoint procedures have the effect of breaking up a long job into a series of small ones. Each portion

of the work is run as a separate and independent part, and each part is checked after it is completed. If the check is correct, enough information is written out to make it possible to return to this last point automatically. If not, the portion of work just completed incorrectly is discarded, and the system restarts from the last point at which the work is known to be correct.

A restart procedure (1) backs up the entire computer system to the specified point in the procedure, usually a checkpoint (tape files are backspaced or rewound; card units and printers are adjusted manually); (2) restores the storage of the computer to its status at the preceding checkpoint (this may include the adjustment of accumulated totals, reloading the program itself, reestablishing switches and counters, restoring constant factors, and so on).

The proper use of checkpoint and restart procedures in a program contributes to the overall operating efficiency of a computer system. If power failure or serious machine malfunction occurs, these procedures provide a means of rerunning only a small part of a job without having to rework an entire job. This may mean a saving of many hours of machine time.

Restart procedures also allow interruption of a given job for the scheduling of other jobs that need immediate or emergency attention. Thus, any procedure may be interrupted intentionally by the operator and replaced with another job when necessary. Provision for restart is also convenient at the end of a shift or other work period when the operation of a job must be terminated without loss of production time. Finally, restart procedures provide interruption of machine operation for emergency repairs or unscheduled maintenance.

Machine Checking

Procedures perform two functions. First, they accomplish useful work; second, they control the quality and accuracy of the work.

In the data processing procedure, useful work consists of such operations as sorting, calculating, collating, reading, and printing. Control operations are necessary to establish and maintain accounting controls, calculation checks, and machine checks. The programmer can use these checking devices at his own discretion, or they can be programmed in an operating system of control programs. Basically, two types of checks may be written:

1. Checks on the validity of data handled by the input/output units.
2. Checks on the handling of data within the computer, including checking for legitimate instruction code, arithmetic overflow, valid signs of numeric quantities, and other check indicators.

In most cases, it is not necessary to halt machine operation when an error condition is detected. The programmer may insert special transfer or branch instructions designed to handle certain types of errors as exceptions. An error in reading a record from tape, for example, may be programmed to backspace the tape and reread the record. If a correct reading is obtained the second time, normal operation continues. If the error persists, operation can be interrupted, or the incorrect record can be noted and operation continued.

In some systems, the error indicator initiates a special routine that places the computer under control of service-mode repair subroutines to accomplish the program repair, and then proceeds back to the interrupted program for continued processing. This operation is completely automatic. In other systems appropriate testing instructions achieve the same end result.

Index

Printed in U. S. A. C20-1684-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]